INGENIERÍA INFORMÁTICA

Universidad Carlos III de Madrid

ESCUELA POLITÉCNICA SUPERIOR
INGENIERÍA INFORMÁTICA

X-Aute

Herramienta para la Automatización de las pruebas en las Interfaces de Usuario Graficas desarrolladas con XAML

Realizado por:

D. Julio César Ayllón Bonet

Dirigido por:

Prof. Dr. Ricardo Colomo Palacios

Departamento de Informática

Diciembre, 2008

RESUMEN

Las pruebas automatizadas sobre interfaces de usuario se están convirtiendo en elementos indispensables para asegurar la calidad de esta parte del software, la cual cada vez está tomando más importancia, de una manera mucho más rápida, efectiva y eficaz que realizando tan sólo testeo manual. Con la automatización de los test las organizaciones pueden ahorrar gran cantidad de tiempo y recursos o producir software de mayor calidad mucho más rápido. Todo esto unido a la llegada de las nuevas interfaces de usuario gráficas *WPF* y a la adopción de estas como estándar para el desarrollo de las nuevas *GUI* en los *Sistemas de Información Corporativos,* hace necesario el desarrollo de una herramienta capaz de asegurar la calidad de dichas *GUI,* mediante un testeo automatizado. Este proyecto presenta una herramienta, *X-Aute*, que se basa en el *framework UI Automation,* a través del cual es capaz de realizar la tan necesaria automatización de pruebas sobre las interfaces de usuario gráficas *WPF*.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

SECCIÓN

I

INTRODUCCIÓN Y OBJETIVOS

1. INTRODUCCIÓN

En este capítulo se realizará una introducción sobre el ámbito de estudio, la presentación del problema asociado a dicho estudio, la descripción de la solución que se ha llevado a cabo y la estructura de toda la memoria de este *Proyecto Fin de Carrera*.

1.1 DESCRIPCIÓN DEL ÁMBITO DE ESTUDIO

Las interfaces de usuario gráficas son los artefactos tecnológicos de los sistemas interactivos que posibilitan la interacción amigable de los usuarios con la aplicación.

La interfaz gráfica de usuario es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y las acciones disponibles en la interfaz. Habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora.

Las interfaces gráficas surgen de la necesidad de hacer los ordenadores más accesibles para el uso de los usuarios comunes, pues hasta su aparición había que poseer grandes conocimientos para realizar cualquier tarea con un ordenador. Esta limitación fue salvada gracias al desarrollo de los entornos gráficos, que permitieron que las personas pudieran acceder a un ordenador sin tener que pasar por el tortuoso proceso de tener que aprender a manejar un entorno bajo línea de comandos.

La historia reciente de la informática está indisolublemente unida a las interfaces gráficas, puesto que los sistemas operativos gráficos han ocasionado grandes consecuencias en la industria del software y del hardware.

Tradicionalmente los ejecutivos, a los que estaban destinados la mayoría de los Sistemas de Información Corporativos, eran bastantes reticentes a utilizarlos debido a las peculiaridades de la información que tenían que manejar. Pero la evolución de las posibilidades técnicas de los ordenadores y la aparición de sistemas fáciles de aprender y de usar, basados en interfaces gráficas intuitivas y amigables, han conducido a una demanda cada vez mayor de Sistemas de Información a la medida de ejecutivos que les faciliten el desarrollo de sus tareas.

En este contexto cada vez es más importante desarrollar sistemas y estándares de aseguramiento de calidad orientados a garantizar el funcionamiento correcto de la interfaz de usuario. Para ello es necesario el desarrollo de pruebas de usabilidad con usuarios finales, planes y casos de test que cubran todas las posibilidades de interacción entre el usuario y la interfaz, y herramientas de automatización capaces de ejecutar los casos de uso más frecuentes en distintas condiciones, ante cambios en el desarrollo o sobre diferentes entornos.

1.2 PROBLEMÁTICA

Cualquier desarrollador con experiencia en la implementación de aplicaciones conoce de sobra el esfuerzo que supone probar correctamente una aplicación. Crear casos de prueba, ejecutarlos y analizar sus resultados es una tarea tediosa. Además, es habitual que los requisitos de la aplicación varíen constantemente, con el consiguiente aumento del número de versiones de la aplicación y la refactorización continua del código. En este contexto, es muy probable que aparezcan nuevos errores.

Este es el motivo por el que la automatización de pruebas es una recomendación, aunque no una obligación, útil para crear un entorno de desarrollo satisfactorio. Los conjuntos de casos de prueba garantizan que la aplicación hace lo que se supone que debe hacer. Incluso cuando el código interno de la aplicación cambia constantemente, las pruebas automatizadas permiten garantizar que los cambios no introducen incompatibilidades en el funcionamiento de la aplicación. Además, este tipo de pruebas obligan a los programadores a crear pruebas en un formato estandarizado y muy rígido que pueda ser procesado por un framework de pruebas.

En ocasiones, las pruebas automatizadas pueden reemplazar la documentación técnica de la aplicación, ya que ilustran de forma clara su funcionamiento. Un buen conjunto de pruebas muestra la salida que la aplicación produce para una serie de entradas, por lo que se puede considerar suficiente para entender el propósito de cada funcionalidad.

El problema que se plantea en la automatización de pruebas de las interfaces gráficas es su gran dificultad, debido a la propia naturaleza de las GUI. Son la capa más cercana al usuario y es la que más modificaciones sufre, su código es complejo, el estado de la interfaz de usuario puede variar entre las pruebas mientras que el estado de la funcionalidad subyacente permanece estable, etc.

Debido al poco tiempo transcurrido desde la aparición del *WPF,* a día de hoy no existen muchas herramientas capaces de realizar un testeo automatizado de pruebas sobre interfaces *WPF* con eficacia y completitud.

1.3 DELIMITACIÓN DE LA SOLUCIÓN

En este proyecto se va a abordar el diseño de *X-Aute, una herramienta para la automatización de las pruebas en las interfaces de usuario gráficas desarrolladas con XAML.*

De forma concreta, *X-Aute* es una herramienta que permite realizar los scripts de pruebas automatizadas para las interfaces de usuario gráficas de un modo muy sencillo por muy complejas que dichas interfaces puedan llegar a ser.

También se pretende que la herramienta sea capaz de ejecutar los test para manipular automáticamente la interfaz gráfica sin la necesidad de ninguna interacción por parte del usuario, con lo que se consigue eliminar las probabilidades de error que este pueda cometer y se pueden realizar tantas veces como se quieran las mismas pruebas con la seguridad que las entradas serán siempre las mismas.

1.4 ESTRUCTURA DE LA MEMORIA

El resto de esta memoria está estructurada de la manera que a continuación se describe. En primer lugar, el capítulo 2 plantea los objetivos que se establecen ante la realización de este proyecto. La Sección II engloba los capítulos 3, 4, 5 y 6, en los que se analizan las tres grandes áreas que se encuentran detrás de la realización de este proyecto y que son: *los sistemas de información comerciales en las organizaciones*, *XAML el nuevo lenguaje para el desarrollo de interfaces de usuario gráficas en WPF y la automatización de pruebas en interfaces de usuario gráficas*. En primer lugar se estudia cada una de ellas por separado para, al final, poder establecer la relación existente entre ellas de manera que permita establecer una base sólida sobre la que fundamentar el resto del proyecto.

La Sección III, que comprende los capítulos 7, 8, 9, 10 y 11 aborda la descripción de las características y el proceso de desarrollo e implementación de la solución diseñada en este proyecto. Para ello, en primer lugar se presentan las principales funcionalidades proporcionadas por la solución, después se hace una descripción de las herramientas empleadas durante el proceso de construcción, se detalla la metodología de desarrollo aplicada, se expone cual es el calendario de trabajo y el presupuesto del proyecto y por último se muestran los detalles de la implementación de la herramientas *X-Aute*.

La Sección IV recoge en los capítulos 12 y 13 las conclusiones obtenidas tras la realización de este proyecto así como las posibles vías de trabajo que se pueden abrir tomando como base el presente *Proyecto Fin de Carrera*.

Por último, se incluye en forma de apéndice el documento técnico fruto de la fase RU de la metodología de desarrollo aplicada, el *Documento de Requisitos de Usuario.*

2. OBJETIVOS

El objetivo principal de este proyecto es el desarrollo de una herramienta que permita la *automatización de las pruebas en las interfaces de usuario gráficas* desarrolladas mediante la nueva tecnología que *Microsoft®* aporta con el *.Net Framework 3.0®*. Dicha tecnología es *Windows Presentation Foundation® (WPF)* a la que acompaña un nuevo lenguaje descriptivo de interfaces gráficas, *XAML*. Se ha llevado a cabo el desarrollo de una solución donde se muestra el potencial innovador aportado por *Microsoft®* de cara al testeo automatizado para las novedosas y potentes interfaces desarrolladas mediante *WPF*.

Se desea que la herramienta permita crear los scripts de pruebas automatizadas, pero que lo haga de un modo muy sencillo sin exigir al usuario que tenga unos conocimientos técnicos muy elevados. Finalmente la herramienta deberá poder ejecutar los scripts de pruebas automatizadas sin más interacción con el usuario que lanzar la batería. Con esto se consigue eliminar por completo las probabilidades de error por parte del usuario, al mismo tiempo que posibilita la repetición del test en el momento deseado con exactamente las mismas características, para pruebas de regresión, etc.

Por otra parte se analiza como cada vez más las organizaciones dependen de sus sistemas de información, los cuales pueden ofrecerlas grandes ventajas competitivas. Se ha estudidado la necesidad de que tales sistemas sean aplicaciones software de calidad, objetivo para el cual lo test automatizados son de una gran utilidad.

Para lograr los objetivos planteados y como paso previo al desarrollo de la herramienta de testeo automatizado sobre interfaces gráficas *WPF*, se ha de llevar a cabo un estudio del área de trabajo, teniendo en cuenta las particularidades de los sistemas de información, de las nuevas interfaces de usuario gráficas elaboradas con *XAML* y *WPF* y de la automatización de pruebas. Para el desarrollo de la herramienta será conveniente aplicar técnicas y metodologías de desarrollo que permitan una mejor gestión del proceso y que aseguren la construcción de una aplicación de calidad.

Y como conclusión, no hay que olvidar el carácter académico de este trabajo, por lo que su realización debería de servir de elemento integrador de todos los conocimientos y competencias que han sido adquiridos durante el desarrollo de la titulación de **Ingeniería Informática Superior** en la Universidad Carlos III de Madrid.

SECCIÓN

II

ESTADO DEL ARTE

3. SISTEMAS DE INFORMACIÓN PARA LA EMPRESA

Cada día más, las empresas dependen en mayor medida de la *información*, por ello resulta muy provechoso diseñar e implantar *Sistemas de Información (SI)* que produzcan y gestionen dicha *información*, con el objetivo de asegurar que ésta sea fiable, exacta y esté disponible cuando se la necesite y por quien la necesite para tomar una decisión. Por tanto, los *Sistemas de Información* pueden ofrecer a las organizaciones grandes ventajas competitivas además de tener un impacto estratégico reduciendo los costes de producción y permitiendo identificar potenciales segmentos de mercado.

3.1 LA INFORMACIÓN

Antes de comenzar a profundizar en las características de los *Sistemas de Información* y el modo en que gestionan la *información* se ha de tener una idea bien clara y concisa de que es exactamente la *información*.

De forma habitual se utilizan los conceptos *información* y *datos* indistintamente, empleándolos como sinónimos, pero sin embargo no lo son, puesto que los propios *datos* son la materia prima de la *información*, a saber:

Un ***dato*** es un elemento de conocimiento que carece de significado por sí mismo o que está fuera de su contexto. Se trata de algo incompleto que necesita otro dato o un proceso de elaboración que le dé más sentido (De Pablos Heredero, Izquierdo Loyola, López-Hermoso, Martín-Romo Romero, Montero Navarro, & Nájera Sánchez, 2001).

Por ***información*** se entiende un *dato* o *conjunto de datos*, elaborado y situado en un contexto, de forma que tenga un significado para alguien en un momento y lugar determinados. (De Pablos Heredero, Izquierdo Loyola, López-Hermoso, Martín-Romo Romero, Montero Navarro, & Nájera Sánchez, 2001).

Por tanto el *dato* tiene un carácter individualizado y simple frente a un producto elaborado y contextualizado como es la *información*.

Figura 3.1. *Transformación de datos en información* **(Applegate, McFarlan, & McKenney, 1996)**.

Los *Sistemas de Información* convierten los *datos* en *información*, ayudando además a gestionarla y a utilizarla de modo que sea rentable para la empresa. El producto que se ha generado se utilizará como soporte en las acciones o decisiones que se vayan a tomar y su valor depende en la medida en que afecte a tales hechos.

Existen diversos tipos de *información* y distintas maneras de categorizarlos (De Pablos Heredero, Izquierdo Loyola, López-Hermoso, Martín-Romo Romero, Montero Navarro, & Nájera Sánchez, 2001), algunas clasificaciones son:

- ***Información interna y externa***: Representan los ámbitos en lo que se genera la *información*. Por tanto, la producida en el interior de la empresa, como consecuencia de distintas actividades, es la *información interna,* mientras que la *información externa* es aquella que se genera en el exterior en el que se sitúa la organización y desarrolla su actividad.

- ***Información de gobierno y de consumo***. La primera es aquella que hace referencia a los objetivos y normas que rigen las decisiones a adoptar. La *información de consumo* establece el estado de las cosas. Refleja la situación de los hechos que acontecen en la organización.

3.2 GESTIÓN DE LA INFORMACIÓN

La *información* es hoy en día uno de los recursos más importantes de las organizaciones y de su correcta gestión se pueden sacar múltiples beneficios para la compañía.

Una vez los *datos* son transformados en *información,* ésta es utilizable para la administración empresarial, siendo su base fundamental de éxito. Para conseguir dicho éxito es necesario *gestionar la información* de un modo adecuado y enfocado al objetivo específico para cada caso. Por tanto se desarrollan e implantan *Sistemas de Información* que la gestionen, obteniendo un uso adecuado de la *información,* lo que permite mejorar la planificación empresarial, la toma de decisiones y los resultados (Adeoti-Adekeye, 1997).

La *gestión de información* es la capacidad transversal de una organización para crear, mantener, capturar y hacer disponible la *información* para la toma de decisiones en el lugar correcto y en el momento adecuado por parte de quien mejor la pueda utilizar y con el mínimo coste posible (Langemo, 1980).

El aspecto clave para una correcta *gestión de la información* es el tratamiento de la *información* a través de tecnologías modernas (Adeoti-Adekeye, 1997).

3.3 SISTEMAS DE INFORMACIÓN

Al igual que se hizo con el concepto de *información*, se dará una definición para el término *sistema*. Un ***sistema*** es un conjunto de elementos en interacción dinámica organizados para la consecución de un objetivo (De Pablos Heredero, Izquierdo Loyola, López-Hermoso, Martín-Romo Romero, Montero Navarro, & Nájera Sánchez, 2001).

Una vez definidos ambos conceptos, es el turno de precisar que es un *Sistema de Información*. Un ***Sistema de Información*** es un conjunto de personas, procedimientos, bases de datos, hardware y software que de forma coordinada reúne, procesa, almacena y distribuye datos, para el procesamiento de transacciones en el nivel de operaciones, e información, para apoyar la toma de decisiones (Duff & Assad, 1980). Los *SI* proporcionan soporte a la dirección como también al resto de trabajadores ayudando a analizar problemas y a visualizar situaciones complejas, además de ayudar en la coordinación de la organización (Laudon & Laudon, Management Information Systems: New Approaches to Organization and Technology, 1998).

En resumen, un *Sistema de Información* es un conjunto de elementos que interactúan entre sí con el fin de apoyar las actividades de una organización ayudándola a lograr sus objetivos.

Otra posible definición es: El *Sistema de Información* empresarial constituye el conjunto de recursos, componentes y medios de comunicación de la empresa que sirven como soporte para el proceso básico de transformación de la información (De Pablos Heredero, Izquierdo Loyola, López-Hermoso, Martín-Romo Romero, Montero Navarro, & Nájera Sánchez, 2001). Se puede ver, por tanto, como un sistema que toma información como materia prima y a través de varios procesos de

transformación la convierte en información elaborada. Para ello consta de los siguientes elementos funcionales, que están relacionados con la organización y su entorno: (Adeoti-Adekeye, 1997):

- ***Percepción:*** Introducción de información en la organización, tanto capturada como generada.
- ***Registro:*** Captura física de datos.
- ***Procesamiento:*** Transformación de acuerdo a las necesidades específicas de la organización.
- ***Transmisión:*** Los flujos que tienen lugar en un sistema de información.
- ***Almacenamiento:*** Porque se supone que la información será utilizada en un futuro.
- ***Recuperación:*** Búsqueda de la información almacenada.
- ***Presentación:*** Creación de informes y difusión.

Para que un *Sistema de Información* empresarial logre alcanzar sus objetivos, ha de ejecutar tres actividades básicas claramente diferenciadas: *entrada, procesamiento y salida*. La *entrada* es el proceso mediante el cual el *SI* captura *datos* en bruto de fuentes internas o externas a la organización para luego poder procesarlos. El *procesamiento* es la capacidad del *SI* para efectuar cálculos de acuerdo con una secuencia de operaciones preestablecidas. Supone la manipulación, conversión y análisis de los *datos* de *entrada* para darles sentido, convirtiéndolos en *información* útil de modo que posteriormente pueda ser utilizada de forma adecuada, sin errores y con el menor coste posible. Finalmente la *salida* es la capacidad de un *SI* para distribuir la *información* procesada y elaborada de modo que este accesible para las personas que necesiten utilizarla.

Además de las anteriores existe otro elemento fundamental en los *SI*, la *retroalimentación*, por medio de la cual las salidas se devuelven a los miembros apropiados de la organización para ayudarles a controlar el funcionamiento de la etapa de entrada (Laudon & J.P., Sistemas de Información gerencial: Organización y Tecnología de la empresa conectada a la red, 2002).

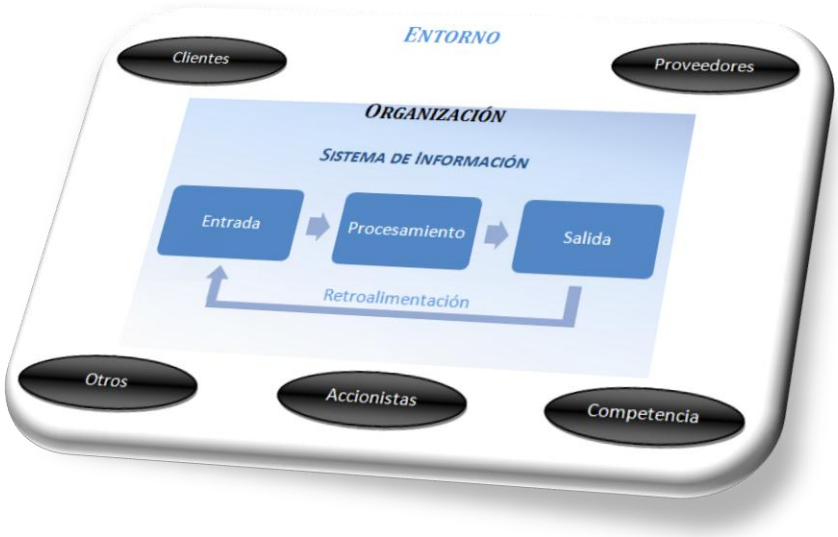


Figura 3.2. *Actividades de un Sistema de Información.*

3.4 CLASIFICACIÓN DE LOS SISTEMAS DE INFORMACIÓN EN LA EMPRESA

La toma de decisiones es el proceso que más caracteriza la actividad de un directivo. Para poder tomar la decisión correcta en cada momento precisa una *información* de calidad y adecuada al tipo de medida que haya que tomar. La aplicación de los *SI* es la de ayudar a los directivos a tomar decisiones. Dependiendo del tipo de decisión a tomar, los *SI* existentes, pueden agruparse en distintos niveles.

Antes de comentar cuales son los distintos tipos de *SI* se han de exponer los tres niveles de decisión y gestión presentes en las compañías.

Las organizaciones están estructuradas mediante una jerarquía, de este modo y gracias a un conjunto de procedimientos formales coordinan el trabajo que cada nivel de la organización ha de realizar. Normalmente dicha jerarquía posee una estructura de pirámide en la que tanto la autoridad como la responsabilidad del personal aumentan con la altura. Por tanto, en los niveles superiores de la jerarquía piramidal aparecen los empleados de dirección mientras que los niveles inferiores están formados por el personal operativo. Cada nivel de la organización tiene unos objetivos diferentes al resto de niveles, lo que crea distintos intereses y puntos de vista, implicando que cada altura de la pirámide tome diferentes tipos de decisiones. Como resultado de todo esto, en toda organización pueden distinguirse tres niveles diferentes de decisión y gestión (De Pablos Heredero, Izquierdo Loyola, López-Hermoso, Martín-Romo Romero, Montero Navarro, & Nájera Sánchez, 2001):

- ***Nivel de la dirección estratégica:*** Formado por la alta dirección. Su función más distintiva es la elaboración de la estrategia (definir objetivos y planes a largo plazo) que ha de seguir la organización. Fundamentalmente la *información* manejada por los directivos de

este nivel es *externa* a la empresa, siendo habitual el uso de la intuición en los procesos de toma de decisiones, las cuales no son nada estructuradas.

- ***Nivel de la dirección táctica:*** Su principal cometido es la planificación táctica, es decir, formular los planes a medio plazo, orientando y controlando que tales planes conduzcan a la consecución de los objetivos establecidos por el nivel estratégico. Las decisiones que se tomarán en este nivel son una combinación de decisiones estructuradas y no estructuradas, basándose generalmente en *información interna*.

- ***Nivel de la dirección operativa:*** Lo forman los directivos responsables de la programación y control de las operaciones básicas de la empresa. La *información* que aquí se maneja es la relacionada con la planificación y el seguimiento de las actividades elementales de la compañía, referidas a un corto período de tiempo. Las decisiones tomadas por este nivel de responsabilidad son de tipo estructurado.

En función de las necesidades de *información* de cada uno de los niveles de decisión y gestión, y atendiendo al problema que tratan de resolver se puede establecer la siguiente clasificación de los *Sistemas de Información:*

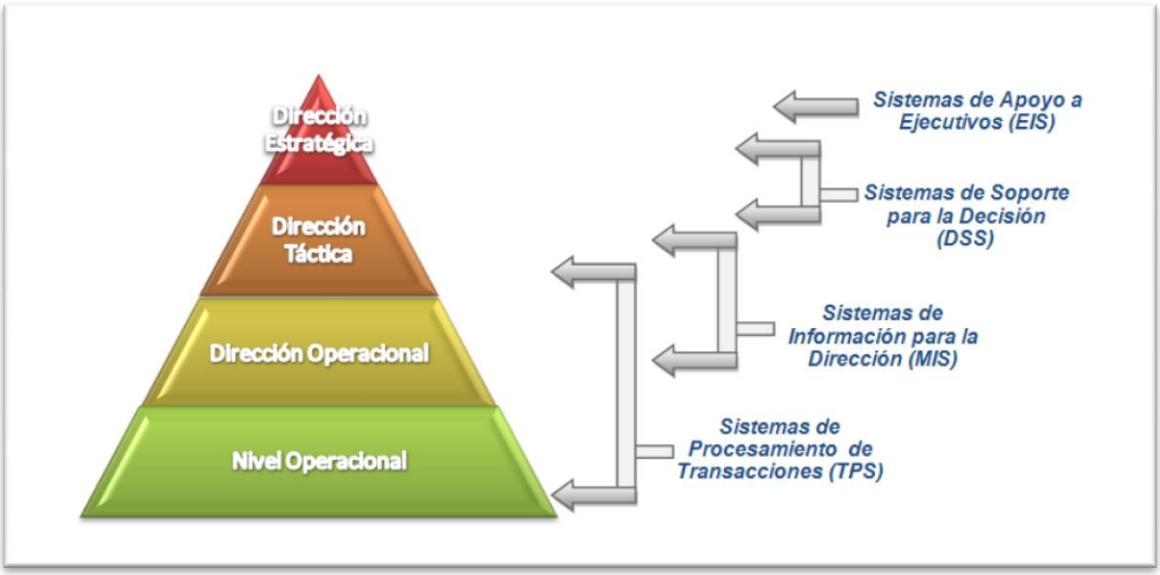


Figura 3.3. *Clasificación de los SI en la empresa (I).*

3.4.1 Sistemas de Información del Nivel Operativo

En correspondencia con este nivel, se encuentran los *Sistemas de Procesamiento de Transacciones, TPS*, diseñados para procesar la gran cantidad de datos que se generan en las

operaciones de las empresas. Los objetivos que persiguen este tipo de *SI* son los de aumentar la productividad y capturar los datos relativos a las transacciones realizadas por la organización, que sirve de base para las tareas de control que realizan los mandos del nivel operativo.

3.4.2 Sistemas de Información del Nivel Táctico

Los *Sistemas de Información para la Dirección, MIS*, fueron concebidos como respuesta a las necesidades asociadas a la toma de decisiones por parte del nivel intermedio de la dirección. Estos sistemas operan sobre la *información* generada por los *TPS* en forma de resúmenes, consolidaciones, etc. La *información* que producen es de tipo *interno*, estructurada y en formatos predeterminados. Así pues, la función principal de estos sistemas es la de proporcionar al directivo del nivel medio los elementos necesarios para desarrollar su trabajo, consistente en una toma de decisiones relativamente programada.

3.4.3 Sistemas de Información del Nivel Estratégico

En este nivel aparecen los *DSS, Sistemas de Soporte para la Decisión.* Según Parker y Case (1993) los *DSS* se definen como un sistema que proporcionan a los directivos herramientas que les apoyen en la toma de decisiones poco estructuradas, de manera más o menos personalizada. Los *DSS* no pretenden resolver los problemas por sí mismos, sino ser una herramienta auxiliar, capaz de proporcionar la *información* necesaria para resolverlos. Estos *sistemas* utilizan la *información* generada por los *TPS* y los *MIS* además de otra *información* que obtiene de fuentes *externas* para facilitar la toma de decisiones complejas y rápidas.

Un tipo concreto de *DDS* son los *EIS, Sistemas de Apoyo a Ejecutivos.* Son los *sistemas* que proporcionan las herramientas e *información* necesaria para la toma de decisiones en las más altas instancias de la empresa, las decisiones estratégicas. Para ello tienen en cuenta *datos externos* como cambios en la legislación fiscal o comercial vigente y los aglutinan con la *información* extraída de los *MIS* y los *DSS* internos.

Tradicionalmente los ejecutivos, a los que están destinados los *EIS,* eran bastantes reticentes a utilizar *SI* debido a las peculiaridades de la *información* que tenían que manejar. Pero la evolución de las posibilidades técnicas de los ordenadores y la aparición de ejecutivos familiarizados con las tecnologías automáticas han conducido a una demanda, cada vez mayor, de *Sistemas de Información* a la medida de ejecutivos que les faciliten el desarrollo de sus tareas. Por todo ello, algunas de las características que presentan los *EIS* son (Applegate, McFarlan, & McKenney, 1996):

- Sistemas fáciles de aprender y de usar, basados en interfaces gráficas intuitivas y amigables.
- Sistemas adaptables y personalizables, de modo que faciliten la tarea habitual del ejecutivo.
- Sistemas alimentados mediante datos externos.

3.5 OTROS SISTEMAS DE INFORMACIÓN

Por último, para completar la clasificación descrita, es necesario considerar otros dos tipos de sistemas de diferente localización al resto.

Encontramos, por una parte, los llamados *Sistemas Ofimáticos, OIS.* Son sistemas que realizan un tratamiento automatizado de la *información* facilitando algunas tareas de las áreas más habituales en un entorno de oficina, por ejemplo elaboración de documentos habituales o la gestión de una agenda de trabajo personal. Aparentemente se podrían situar en el nivel ocupado por determinado personal de *staff* en la empresa, pero prácticamente todos los integrantes de ciertas empresas utilizan algunos componentes de estos *sistemas*. Por tanto, no pueden situarse en ninguna parte de la pirámide organizativa.

De otro lado, los *Sistemas Expertos, SE*, que son utilizados para la resolución de problemas complejos relativos a determinadas partes de la organización. Este tipo de *sistemas* a menudo responden a las características de los sistemas que se utilizan en la toma de decisiones estratégicas, pero no los podemos situar en dicho nivel. Puesto que los *SE no* suelen, y en general no pueden, ser utilizados para la toma de decisiones no estructuradas, puesto que cada una de estas decisiones es única y no existe experiencia previa que sirva de apoyo en la construcción del *sistema*. Más bien son utilizados en la toma de decisiones relativamente estructuradas, generalmente de carácter táctico u operativo, aunque de gran complejidad. Un *SE,* no puede ser considerado un *SI* en rigor, sino más bien, una herramienta que puede pertenecer o no al mismo.

3.6 ANÁLISIS COSTE-BENEFICIO DE LOS SI EN LA EMPRESA

El desarrollo e implantación de *Sistemas de Información* en una organización es una inversión y como tal es susceptible de valoración a lo largo del tiempo. Es necesario hacer un análisis comparativo para comprobar si los beneficios justifican los costes.

Habitualmente para realizar un análisis coste-beneficio se suele acudir a técnicas convencionales de valoración de inversiones, como la *tasa interna de retorno (TIR), valor actualizado neto (VAN), el payback,* etc. Normalmente resulta muy complicado aplicar estas técnicas debido a la dificultad de calcular los beneficios a priori. Los costes han de estimarse en la fase de viabilidad, pues en esta etapa son más cuantificables, y revisarse en las fases posteriores.

Los tipos de costes de un proyecto de desarrollo e implantación de un *SI* que merecen la pena ser destacados son dos:

3.6.1 Costes Económicos

- ***Análisis de sistemas y diseño:*** Se debe considerar el coste total del proyecto.
- ***Compra del hardware:*** Se recomienda barajar alternativas como leasing o renting.
- ***Compra del software:*** Son los más difíciles de estimar.
- ***Costes de formación:*** Hay que formar el personal para utilizar el sistema.
- ***Costes de instalación:*** Puede ser significativo si es necesario construir algo, etc.
- ***Costes de conversión y cambio:*** Relacionados con el almacenamiento de los datos del viejo sistema al nuevo, de una forma segura.
- ***Costes de oportunidad:*** Si se pretende reemplazar a las personas con máquinas, habrá que pagar doblemente.

3.6.2 Costes Operativos

- ***Mantenimiento de hardware y software***.
- ***Costes de electricidad, papel, etc***.
- ***Costes asociados con el personal para operar el sistema:*** Por ejemplo un centro de informática, empleados para la introducción de datos.

Realizar un análisis coste-beneficio del desarrollo e implantación de un *Sistema de Información* en la empresa, supone evaluar la optimización del negocio por el uso del *SI* en concreto. En caso de que el resultado sea positivo será debido, fundamentalmente, a que se ha realizado una óptima gestión de costes, pero también, a un aumento del valor producido.

Se han expresado cuales son los costes más importantes a la hora de implantar un *SI* en la empresa así como también los motivos causantes de que el resultado del análisis-coste beneficio sea positivo, pero no se ha dicho nada acerca de cómo realizar un buen presupuesto, a continuación se indica.

Saber si el resultado del análisis es positivo o no es muy interesante para la empresa, pero para ello es necesario realizar un buen cálculo de los beneficios, que como se ha comentado es algo muy difícil de hacer a priori. Por ello, otra medida que toman las empresas es la de calcular el impacto económico que la implantación del *SI* tendrá sobre la organización. Para la realización de estos presupuestos existen múltiples directrices que se pueden seguir, pero se ha comprobado que en

muchas ocasiones los cálculos que indican no son realistas y obligan a las organizaciones a aceptar soluciones que distan mucho de ser las óptimas (Barreau, 2001).

Una de las maneras más adecuadas de valorar el impacto causado por la introducción de un *SI* en una empresa es la llamada metodología del *análisis centrado en el trabajo*. En ella se evalúa un *SI* en el marco de la organización teniendo en cuenta quien y como lo usará, la estructura de la *información* que manejará y los procesos que se llevarán a cabo en tal sistema. La aplicación de esta metodología permite identificar las siguientes categorías, que normalmente se suelen omitir en los presupuestos:

- ***El salario, los gastos del personal y la dirección implicados en el análisis e implantación del sistema:*** Es muy habitual que los directores presupuesten correctamente las remuneraciones y el tiempo de los empleados que componen el equipo de trabajo, pero que se olviden de tener en cuenta el esfuerzo propio que invierten en tareas como revisión, análisis y toma de decisiones relacionadas con el proyecto.

- ***Los costes relacionados con otras fuentes y trabajos que se ven afectados como consecuencia del proceso de implantación:*** Si el personal está muy involucrado con el establecimiento del sistema, la calidad de otras servicios puede verse afectada temporalmente. Para evitar esta disminución de la calidad puede ser necesario la contratación temporal de nuevos empleados, siendo este un detalle que se suele pasar por alto en los presupuestos.

- ***Los costes de la modificaciones en las instalaciones donde implantar el SI:*** Puede ser necesario cambiar el mobiliario, fontanería, cableado, etc.
- ***Los costes y el período de formación:*** Normalmente cuando se compra un *SI,* este trae consigo un paquete de formación durante un plazo de tiempo. Pero una vez ha vencido el período de tiempo establecido puede ser necesario contratar a más personal para que siga impartiendo la formación. Además el personal de la organización sufre altas y bajas lo que hace que la formación pueda convertirse en un proceso continuo. Otro coste se puede producir al querer modificar particularmente el sistema contratado para lo que será necesario contratar nuevamente más personal.

- ***Los costes de migración y conversión de los sistemas y procesos existentes:*** La adquisición de un nuevo *SI* puede implicar la conversión de datos y la reingeniería de procesos para adaptarse a la nueva situación. Los costes de conversión deben abarcar, si es necesario, el desarrollo de procedimientos para la transferencia de los datos al nuevo sistema, las pruebas y el borrado de los datos antiguos.

Para intentar minimizar el impacto de los costes ocultos anteriores es posible utilizar cuatro estrategias:

1. Realizar un análisis de la idoneidad del *SI* y los procesos de la organización.
2. Conseguir la implicación del mayor número de directivos posible en el proyecto.
3. Consultar ideas a otros usuarios.
4. Diseñar planes de contingencia para las peores situaciones que se puedan dar.

3.7 RESUMEN

La información se ha convertido en uno de los bienes indispensables y más valiosos de las organizaciones. Su correcta gestión posibilita su óptima utilización, de la que se pueden obtener múltiples beneficios como ventajas competitivas frente a otras empresas, la capacidad para elaborar una mejor planificación empresarial, ayuda para tomar mejores decisiones estratégicas que produzcan una reducción de los costes y la posibilidad de identificar potenciales segmentos de mercado. Todo esto es lo que ofrecen los *Sistemas de Información (SI)* a las organizaciones, siendo ellos los que transforman los datos en información útil para la empresa, que además se encargan de gestionar.

Las organizaciones están estructuradas entorno a una jerarquía que normalmente tiene forma piramidal, donde, en la parte más alta se encuentran los más altos cargos de la empresa y en la parte inferior los empleados del nivel operativo. Cada uno de estos niveles de decisión y gestión tiene unas necesidades diferentes en cuando a la información se refiere, y por tanto, se pueden distinguir tres tipos de *Sistemas de Información*, los *SI* del nivel operativo, los del nivel táctico y los del nivel estratégico. En la *tabla 2.1* se pueden observar algunas de las características de los *SI* correspondientes a cada nivel.

Nivel de Decisión	*Sistemas de Información*	*Tipo de Decisión*	*Origen de la Información*
Nivel Operativo	*TPS*: Sistemas de proceso de transacciones.	Estructuradas	Interna
Nivel Táctico	*MIS*: Sistemas de Información para la Dirección.	Estructuradas y no estructuradas	Generalmente Interna
Nivel Estratégico	*DSS*: Sistemas de Soporte para la Decisión *EIS*: Sistemas de Información para la Alta Dirección.	No estructuradas	Externa

Tabla 3.1. *Clasificación de los SI.*

Además de estos, existen otros dos tipos de *SI* que no se pueden situar en un nivel concreto de la jerarquía como son los *OIS*, o *Sistemas Ofimáticos*, los cuales se utilizan en todos los niveles, y los *SE, o Sistemas Expertos*, que realmente son una herramienta propia de un *SI* más que uno en sí mismo y suelen utilizarse en la toma de decisiones tácticas u operativas.

Figura 3.4. *Clasificación de los SI en la empresa.*

A la hora de desarrollar e implantar un *SI* en una empresa se debe realizar algún tipo de valoración que indique si la inversión va a resultar productiva, lo cual puede hacerse mediante un análisis coste-beneficio. Otra medida que se puede tomar es la de calcular el impacto económico que tendrá la implantación del *SI* en la organización mediante la metodología del análisis centrado en el trabajo.

Los *Sistemas de Información* ayudan, además de lo comentado al comienzo de este punto, a mejorar los servicios ofrecidos a los clientes y al incremento de la productividad debido a la optimización de los recursos disponibles. En definitiva, los *Sistemas de Información* ayudan a las organizaciones a lograr sus objetivos.

4. XAML

Con la llegada de la versión *3.0 del Framework de .Net*, no sólo aparecen nuevas tecnologías como *Windows Presentation Foundation (WPF)* sino que además aparece un nuevo lenguaje con el objetivo de ayudar en el desarrollo de programas modernos. Se trata de *XAML (eXtensible Application Markup Language)*.

Es un lenguaje declarativo que se basa en *XML* y que está pensado para desarrollar interfaces en *WPF*. *XAML* permite separar la interfaz de usuario de la parte lógica de la aplicación, utilizando *XAML* para la interfaz de usuario y *C#* o *VisualBasic.Net* para la lógica de los programas.

Antes de comenzar a detallar *XAML*, se debe dar una pequeña explicación de que es el *Framework 3.0 de .Net* así como también que es *WPF*.

4.1 PLATAFORMA .NET

.Net es la arquitectura que pretende conseguir la conectividad absoluta. Esto quiere decir, que no exista la separación entre páginas Web a un lado y las aplicaciones al otro, que Internet no sea el único responsable de la conectividad y que los diversos lenguajes de programación no sean capacidades imposibles de asociar (Panello).

En definitiva, *.Net* es un proyecto de Microsoft © con el que quiere crear una nueva plataforma que permita desarrollar software con transparencia de redes, con independencia de plataforma y con un rápido desarrollo de aplicaciones (Microsoft Corp., 2006).

.Net es básicamente (Panello):

- ***La infraestructura .Net***. Está formada por el *.Net Framework* ©, *Microsoft Visual Studio .Net* ©, *.Net Enterprises Servers* © y *Microsoft Windows .Net* ©.
- ***Los servicios de Internet*** (*.Net Building Block Services* ©). Ofrecen la posibilidad de acceder a través de programas a determinados servicios.
- ***Programas .Net*** que acceden a tales servicios.

Lo más visible de *.Net* es su infraestructura. Se trata de todas las tecnologías que conforman el nuevo entorno que permite a los desarrolladores crear y ejecutar nuevas aplicaciones.

4.2 .NET FRAMEWORK

Se llama *Framework* o entorno de trabajo, a las *Bibliotecas de Clase Base*, (también llamadas *BCL*) y al *Common Language Runtime, (CLR)*.

Las *bibliotecas de clase* son una nueva estructura jerárquica de clases que envuelven diversas funcionalidades y que están disponibles para cualquier lenguaje *.Net* (*Visual C++.Net, Visual Basic.Net, Visual C#.Net, ASP.Net*, etc.) (Panello).

Las funcionalidades que envuelve la *BCL* son la mayoría de las operaciones básicas que se encuentran involucradas en el desarrollo de aplicaciones, como por ejemplo: administración de memoria, manejo de tipos de datos unificado, operaciones aritméticas, etc.

La *Biblioteca de Clases Base* se organiza en torno a tres grupos clave (Microsoft Corp., 2006):

- *ASP.Net y Servicios Web XML*
- *Windows Forms*
- *ADO.Net*

El *CLR* es el verdadero núcleo del *Framework*, es el entorno que usan las aplicaciones escritas en diversos lenguajes en tiempo de ejecución. El *CLR* gestiona la ejecución de cada ejecutable encapsulándolo, separándolo de otros procesos de la máquina. Ofrece una interoperabilidad multi-lenguaje, o lo que es lo mismo, la característica de que cada aplicación escrita en diferentes lenguajes pueda interactuar sin inconvenientes (Panello). Para esto, la herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por *.Net* en un código inter-medio (*MSIL, Microsoft Intermediate Lenguaje*). Para generar dicho código el compilador se basa en el *Common Language Specification (CLS)* que determina las reglas necesarias para crear ese código *MSIL* compatible con el *CLR* (Microsoft Corp., 2006).

Para ejecutarse es necesario un segundo paso, un compilador *JIT* (*Just-In-Time*) es el que genera el código máquina real que se ejecuta en la plataforma del cliente.

De esta forma, con *.Net* se consigue independencia de la plataforma hardware.

La compilación *JIT* la realiza el *CLR* a medida que el programa invoca métodos, el código ejecutable obtenido se almacena en la memoria caché del ordenador, siendo recompilado de nuevo sólo en el caso de producirse algún cambio en el código fuente (Microsoft Corp., 2006).

4.2.1 .NET Framework 3.0

Microsoft .Net Framework 3.0 (también conocido como *WinFX*), es el nuevo modelo de programación para Windows que combina la potencia del Framework 2.0 con cuatro nuevas tecnologías:

- *Windows Presentation Foundation (WPF)*
- *Windows CardSpace*
- *Windows Communication Foundation (WCF)*
- *Windows Workflow Foundation (WF)*

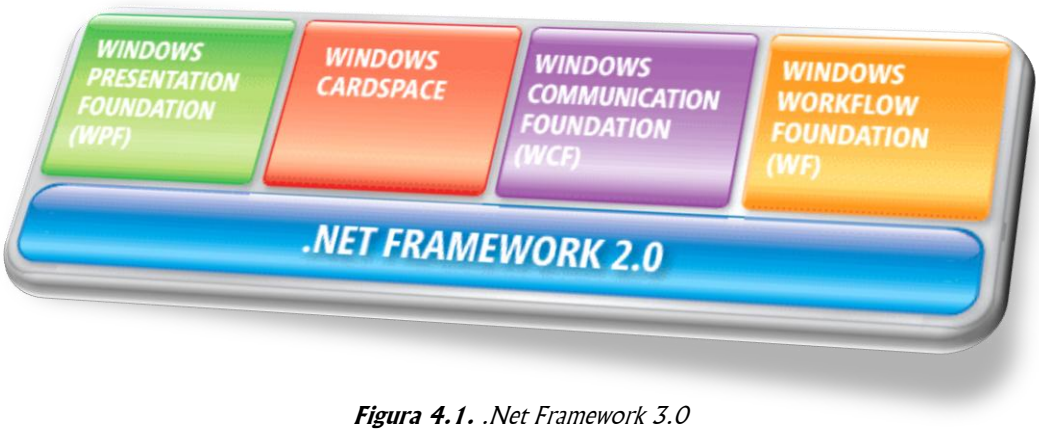


Figura 4.1. *.Net Framework 3.0*

Anteriormente se ha explicado que es el *Framework* y qué componentes lo forman (*CLR* y *BCL*). A continuación se ofrece una breve descripción de las nuevas tecnologías añadidas por el *Framework 3.0*. Puesto que el tema que nos abarca, *XAML*, presenta una gran relación con el *WPF* se expondrá una descripción más detallada de ésta última tecnología.

4.2.1.1 Windows CardSpace

Actualmente todo el mundo posee diversas identidades en Internet (proveedor de correos, sitios de compra, chats, etc.), teniendo que recordar para cada una de ellas el nombre de usuario y la contraseña, además de la dificultad añadida de poner diferentes contraseñas en cada sitio, pues lo contrario puede ser muy peligroso. Es por esto que *Microsoft* © presenta *Windows CardSpace*.

Esta tecnología provee a los usuarios la habilidad de manejar sus identidades digitales de un modo sencillo, seguro y familiar (Microsoft Corporation, 2006).

4.2.1.2 Windows Communication Foundation (WCF)

La aceptación global de los Servicios Web, los cuales incluyen protocolos estándar para la comunicación entre aplicaciones, ha cambiado el desarrollo de software. Por ejemplo las funciones que ahora incluyen los Servicios Web integran la seguridad, la coordinación de transacciones distribuidas y la comunicación fiable. *Windows Communication Foundation* (*WCF*) ha sido diseñado para ofrecer un enfoque manejable de la computación distribuida, además de una gran interoperabilidad y el apoyo directo a la orientación de los servicios. (Microsoft Corporation, 2007)

En definitiva *WCF* es un *Framework* unificado para la construcción segura, confiable, transaccional e interoperable de aplicaciones distribuidas (Microsoft Corporation, 2006).

4.2.1.3 Windows Workflow Foundation (WF)

Es el modelo de programación, el motor y las herramientas que permiten la rápida construcción de aplicaciones "*workflow enabled*". *WF* aumenta radicalmente la capacidad del desarrollador para modelar los procesos de negocio (Microsoft Corporation, 2006). Esta plataforma es altamente flexible y permite el diseño de workflows automatizados u orientados a la interacción humana y puede ser utilizado en una gran variedad de escenarios y productos (Microsoft Corporation, 2005).

La creación de aplicaciones "*workflow enabled*" se divide en las siguientes partes:

- ***Activity Model***: Las actividades son los elementos básicos de los workflow, interpretándolas como una unidad de trabajo que necesita ser ejecutada.
- ***Workflow Designer***: Es el diseñador que permite la composición gráfica de los workflow.
- ***Workflow Runtime***: Es un motor pequeño pero extensible, que ejecuta las actividades que constituyen un workflow. Se puede alojar en cualquier proceso .*Net*, lo que permite a los desarrolladores trasladar cualquier workflow desde una aplicación *Windows Forms* a un web site *ASP.Net* o un Servicio Windows.
- ***Rules Engine***: *WF* posee un motor de reglas declarativas, basado en normas, para el desarrollo de workflows y de aplicaciones .*Net* que utilicen tal flujo.

4.2.1.4 Windows Presentation Foundation (WPF)

Windows Presentation Foundation (WPF), es una de las novedosas tecnologías de Microsoft y uno de los pilares de Windows Vista, aunque su uso no es exclusivo de este Sistema Operativo. *WPF* encierra un trabajo monumental que parte de la gran experiencia acumulada en el desarrollo de interfaces de usuario desde la aparición de *Windows*. *WPF* se apoya en los desarrollos de *Microsoft* para *DirectX* [1] y potencia las capacidades de desarrollo de interfaces de interacción integrando y ampliando las mejores características de las *aplicaciones Windows* y de las *aplicaciones Web* (Katrib, Del Valle, Sierra, & Hernández, 2007).

Habitualmente el personal técnico se preocupa principalmente por la tecnología, los profesionales del software se centran en el modo de funcionamiento de las aplicaciones mientras que el usuario le da una gran importancia a las interfaces. La interfaz de una aplicación constituye una parte fundamental de la experiencia global del usuario con el software particular. En lo que respecta a los usuarios, la aplicación es la experiencia. La experiencia mejorada de los usuarios mediante una interfaz optimizada puede contribuir al incremento de la productividad, a la generación de clientes leales y a una ampliación de las ventas en línea, entre otras muchas.

Los usuarios actuales no se conforman con interfaces de texto, a día de hoy exigen interfaces gráficas, pero además los requisitos de éstas continúan aumentando.

El objetivo de *Windows Presentation Foundation (WPF)* es ayudar a los desarrolladores a crear interfaces de usuario eficaces y atractivas. La plataforma unificada de *WPF* convierte a los desarrolladores en participantes activos en la creación de interfaces de usuario y proporciona un modelo de programación común para aplicaciones independientes. *WPF* permite crear interfaces que incorporen documentos, componentes multimedia, gráficos bidimensionales y tridimensionales, animaciones, características tipo web, etc (Chappell, 2006). Anteriormente para crear interfaces de usuario que ofrecieran todas las características mencionadas eran necesarias varias tecnologías, véase la ***Figura 4.2***.

WPF no reemplaza todas esas tecnologías, puesto que éstas se seguirán utilizando e incluso algunas de ellas pueden interoperar con *WPF,* como por ejemplo *Windows Forms*.

Sin embargo, al proporcionar una amplia gama de funciones en una sola tecnología, se simplifica de forma significativa la creación de interfaces de usuario modernas y más completas.

[1] ***DirectX*** es una colección de API´s (Application Programming Interfaces) creadas por Microsoft que contienen funciones útiles para los programadores de multimedia.

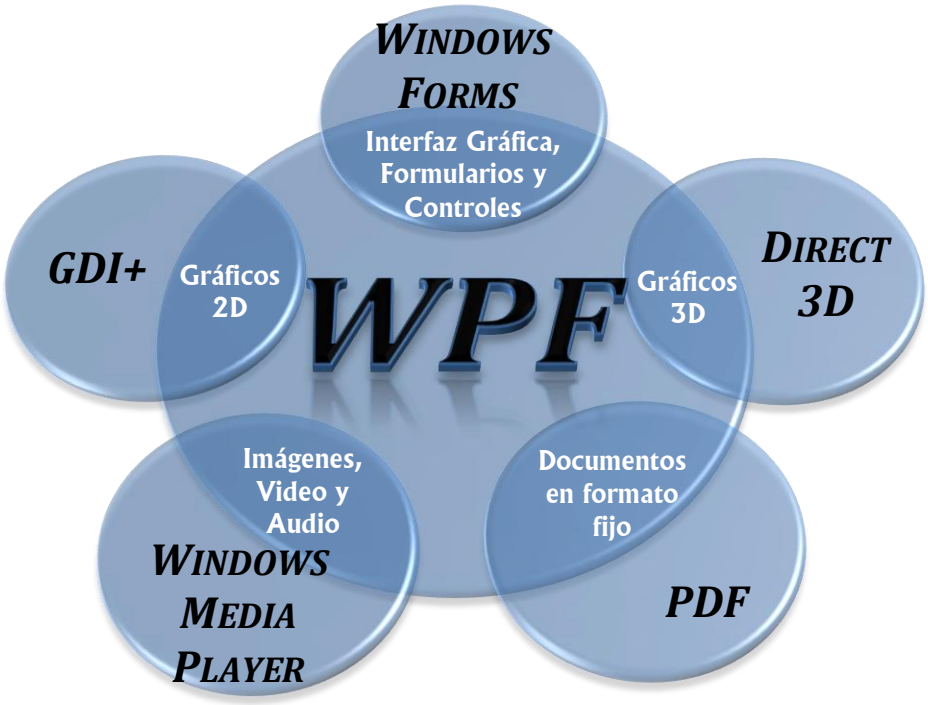


Figura 4.2. *Características de WPF y las anteriores tecnologías.*

WPF es toda una plataforma que da soporte para poder colocar en las interfaces de usuario una amplia riqueza visual e interactiva e integrarlas fácilmente con la lógica de negocio. Los elementos visuales que de las *IU gráficas* podrán tener ahora transparencia, brillo, tonalidades, reflejo, efectos tridimensionales y animaciones. Se podrán desplegar documentos y colocar en las propias aplicaciones audio y video. Además tendrán capacidad de enlace y navegación basado en el modelo de páginas Web. Con *WPF* es posible definir estilos y plantillas propios para propiciar la reutilización, aumentar la productividad de desarrollo y dar una imagen personalizada a las aplicaciones. Además con la arquitectura abierta de *WPF* se permite definir e implementar controles propios y componentes personalizados.

La potencia gráfica de *WPF* se basa en la tecnología DirectX de Microsoft, pudiendo aprovechar al máximo las posibilidades gráficas de que disponga el hardware donde se ejecute la aplicación (Katrib, Del Valle, Sierra, & Hernández, 2007).

Para crear una interfaz de usuario eficaz son necesarios conocimientos que muchos profesionales del software no poseen, lo que implica la necesidad de trabajar con diseñadores de interfaces. Hasta la llegada de *WPF* el trabajo conjunto entre desarrolladores y diseñadores había sido bastante problemático ya que para el desarrollador puede ser muy complicado, o incluso imposible, implementar las, a priori, sencillas ideas del diseñador. Las limitaciones tecnológicas, la

falta de conocimiento, los malentendidos o simplemente los desacuerdos entre ambos pueden ser algunos de los motivos por los que el desarrollador no refleje la visión del diseñador. Por tanto era necesario un método de trabajo mejorado que permitiera la colaboración estrecha de ambos sin que se viese comprometido el nivel de calidad de la interfaz. Con este objetivo *WPF* incluye el lenguaje *XAML*, mediante el cual es posible definir elementos *XML*, como *Button, TextBox, Label*, etc., para especificar exactamente la apariencia de las interfaces de usuario. *XAML* ofrece un método muy sencillo, basado en herramientas, para describir interfaces de usuario y, de este modo, permitir una mejor colaboración entre desarrolladores y diseñadores. En la siguiente imagen se muestra el proceso de colaboración (Chappell, 2006).

Figura 4.3. *Proceso de colaboración entre Diseñador y Desarrollador.*

El diseñador utiliza una herramienta gráfica, como por ejemplo *Expression Interactive Designer ©, Aurora XAML ©,* etc., para definir la apariencia y el modo de interacción de una interfaz de usuario. La herramienta genera automáticamente una descripción de la interfaz en *XAML.* A continuación el desarrollador importa la descripción *XAML* en una herramienta como *Microsoft Visual Studio* © donde escribirá el código de la interfaz, como los controladores de eventos y todas las demás funciones que requiera la aplicación.

Las capacidades de *WPF* son totalmente asequibles desde cualquier lenguaje *.Net,* no obstante, Microsoft propone, un lenguaje declarativo que acopla a la perfección con *WPF, XAML.*

4.3 XAML

Como se acaba de comentar el principal objetivo de *XAML* es describir gráficamente las interfaces de usuario permitiendo trabajar a diseñadores y desarrolladores juntos, pues con la llegada de estas nuevas tecnologías es posible separar la definición de la interfaz gráfica (expresada en *XAML*) de su funcionalidad (escrita en *C#* o *VisualBasic.Net*).

A continuación se explicará más en detalle esta innovadora tecnología que ya, está revolucionando el mundo del desarrollo software.

XAML es el acrónimo de las palabras inglesas *eXtensible Application Markup Language*, Lenguaje Extensible de Marcas para Aplicaciones, en español. Es el lenguaje de ***formato*** de la interfaz de usuario para la *Base de Presentación de Windows (WPF)*.

Es un lenguaje declarativo basado en *XML*, que simplifica en gran medida la creación de *Interfaces de Usuario (IU)* para el *Framework 3.0 de .Net*. Está optimizado para ***describir gráficamente*** interfaces de usuario visualmente ricas (Microsoft Corporation, 2006). Es posible crear elementos visibles de *IU* mediante el marcado declarativo *XAML* y separar la ***definición*** de la ***lógica*** de la aplicación, expresada mediante archivos de código subyacente (también llamados ***code-behind file***). Estos archivos son módulos pre-compilados escritos en alguno de los lenguajes compatibles con el entorno de ejecución de *Microsoft .Net* y están unidos al marcado a través de definiciones de clases parciales (Microsoft Corporation, 2007).

XAML expresa declarativamente la creación de estructuras arbóreas de objetos *.Net.* Con *WPF* como marco de trabajo, estos objetos podrán ser desplegados visualmente e interactuarán con los usuarios permitiendo conformar una rica y apasionante interfaz de aplicación. A la vez se podrán conectar con el resto de la lógica para asociar a esta apariencia de presentación una funcionalidad todo lo compleja que se requiera (Katrib, Del Valle, Sierra, & Hernández, 2007).

Por su naturaleza declarativa, XAML resulta muy apropiado para especificar la interfaz de usuario de una aplicación de modo separado de la lógica de negocio. En este sentido, la combinación de: lenguaje de programación *.Net (C# o VisualBasic .Net), XAML* y *WPF* nos ofrece un adecuado soporte para desarrollar aplicaciones con la arquitectura conocida como ***MVC*** (Modelo-Vista-Controlador). Esta arquitectura consta de tres capas de responsabilidades (Katrib, Del Valle, Sierra, & Hernández, 2007):

- ***Capa Modelo:*** Describe las entidades propias que se encuentran en el dominio del problema que resuelve la aplicación. Son los llamados objetos de negocio. Esta capa será implementada mediante *C#* o *VisualBasic .Net.*

- ***Capa Vista:*** Define la apariencia y funcionalidad de la *IU.* Las enormes capacidades de *WPF* sirven de soporte a esta capa. La especificación de esta capa se hará mediante el lenguaje *XAML.*

- ***Capa Controlador:*** Su propósito es crear un puente, establecer los enlaces entre las capas Modelo y Vista.

Debido a la separación en capas, el patrón *MVC* permite lograr mayor productividad de desarrollo y mayor flexibilidad en el producto.

En su uso típico, los archivos *XAML* son producidos por una herramienta de diseño visual, como por ejemplo *Microsoft Expression* ©. El *XML* resultante es interpretado en forma instantánea por un sub-sistema de despliegue de Windows Vista (concretamente por el *WPF)* que reemplaza al *GDI* de las versiones anteriores de *Windows*. Los elementos de *XAML* se interconectan con objetos de los lenguajes compatibles del entorno ejecución y los atributos de éstos se conectan con propiedades o eventos de tales objetos (Microsoft Corporation, 2006).

Observando el código fuente de un fichero *XAML* se puede ver que no es otra cosa más que un fichero *XML* bien formado, con ciertos espacios de nombres establecidos. Sin embargo la potencia del *XAML* no recae estrictamente en el lenguaje, aunque éste proporcione multitud de ventajas, sino en un conjunto de clases administradas, que permiten en tiempo de compilación, convertir el *XAML* en una clase parcial que contendrá un código equivalente y que posteriormente será utilizada para crear los objetos.

4.3.1 Espacios de nombre

XAML usa el mecanismo de espacios de nombre de XML para representar los espacios de nombre de *.Net*. A un espacio de nombres de *XAML* le puede corresponder más de uno de *.Net,* siempre que no se presenten colisiones.

El primer espacio de nombres de la figura 3.4 incluye todos los tipos que son parte del marco de trabajo de *WPF*. El segundo, al que se le ha asociado el prefijo ***x*** para abreviar cuando se utilice, representa varias utilidades propias de *XAML* y que no lo son de *WPF*. Este último espacio de nombres incluye características que permiten controlar el comportamiento del compilador *XAML* (Katrib, Del Valle, Sierra, & Hernández, 2007).

```
<Window x:Class="WindowsApplication1.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="WindowsApplication1" Height="300" Width="300"
    >
    <Grid>
      ...
    </Grid>
</Window>
```

Figura 4.4. *Ejemplo de Espacios de Nombre XAML.*

4.3.2 Elementos XAML

Como ya se ha comentado, *XAML* es un lenguaje de marcado que sigue las reglas sintácticas de *XML*, donde cada elemento tiene un nombre y puede que varios atributos. Los elementos se

definen uno dentro de otro formando una estructura arbórea, lo que le da expresividad y semántica (Katrib, Del Valle, Sierra, & Hernández, 2007).

XAML tiene un conjunto de reglas que mapean los elementos *XAML* en clases o estructuras de *.Net*, en particular de *WPF*. Los atributos de esos elementos son mapeados en propiedades o eventos de dichas clases y los espacios de nombres *XML* en espacios de nombres *CLR*. Para comprender mejor este punto se muestra un ejemplo que contiene el código necesario para crear un botón (Microsoft Corporation, 2007):

Figura 4.5. *Ejemplo de Elementos XAML.*

En este ejemplo se observan dos elementos *XAML: <StackPanel>* y *<Button>*. Cada uno de ellos es mapeado a una clase definida por el *WPF*. Al especificar la etiqueta de un elemento, se crea una instrucción para el cargador de *XAML* que creará una instancia de la clase correspondiente al elemento cuando el código *XAML* sea compilado o interpretado. Cada instancia será creada invocando al constructor por defecto de la clase y almacenando el resultado. Por tanto, para que un elemento *XAML* pueda ser utilizado la clase a la que se ha de mapear debe ofrecer un constructor por defecto público.

4.3.3 Propiedades en Elementos XAML

Las propiedades de los elementos *XAML* pueden ser establecidas utilizando varias sintaxis. Estableciendo valores a las propiedades se añaden características a los elementos *XAML*. El estado inicial de una clase que ha sido mapeada desde un elemento *XAML* está basada en el comportamiento definido en el constructor por defecto. Normalmente no se quiere utilizar una instancia de clase totalmente por defecto, por tanto se han de añadir propiedades al elemento a mapear para modificar tal comportamiento (Microsoft Corporation, 2007).

4.3.3.1 Sintaxis de Atributos

En *XAML* las propiedades pueden ser expresadas como atributos. La sintaxis de atributos es la más normalizada para el establecimiento de propiedades además de ser la más intuitiva para desarrolladores que han utilizado lenguajes de marcado en el pasado. Por ejemplo en la figura 3.5 se puede apreciar esta sintaxis, se crea un botón y se establece en la propiedad *Content* el valor "Púlsame" (Microsoft Corporation, 2007).

4.3.3.2 *Sintaxis de Elemento-Propiedad*

Existen algunas propiedades de elementos para las que no es posible utilizar la sintaxis de atributos porque el objeto o la información necesaria que se ha de proporcionar como valor de la propiedad no puede ser expresada mediante una simple sentencia. Para estos casos existe una sintaxis diferente, la sintaxis de elemento-propiedad. En este hecho se establece la propiedad del elemento con una nueva instancia del mismo tipo que la propiedad toma como valor. La sintaxis de propiedades se expresa de la siguiente manera *<Elemento.Propiedad>*. Para propiedades que soportan ambas sintaxis, el resultado es el mismo. Sin embargo es preferible utilizar la sintaxis de atributos cuando sea posible pues de este modo se creará un código de marcado más compacto, aunque esto es sólo una cuestión de estilo y no una limitación técnica. En el siguiente ejemplo se establecen las mismas propiedades que en el mostrado anteriormente (Figura 3.5) pero en esta ocasión se hace uso de la sintaxis de elemento-propiedad:

```
<StackPanel>
    <Button>
        <Button.Content>
            Púlsame
        </Button.Content>
    </Button>
</StackPanel>
```

Figura 4.6. *Ejemplo de Sintaxis Elemento-Propiedad.*

Se puede apreciar cómo, para expresar exactamente lo mismo que en el caso anterior, con esta sintaxis se genera un código de mayor tamaño y menos compacto (Microsoft Corporation, 2007).

4.3.3.3 *Propiedades adjuntas*

Acaba de mostrarse dos diferentes maneras de asignar de forma directa un valor a una propiedad en *XAML*. Sin embargo, es posible hacerlo de otro modo, a través de las propiedades adjuntas *(attached property)*. En este caso se le da valor a una propiedad no dentro de la sintaxis del elemento propietario sino desde otro elemento.

```
<Grid>
    ...
    <Button Grid.Row="1">Púlsame</Button>
</Grid>
```

Figura 4.7. *Ejemplo de Propiedades Adjuntas.*

Dentro del botón se hace referencia a *Grid.Row,* es decir, a la propiedad *Row* del elemento *Grid* dándole como valor la cadena *"1"*. *XAML* interpreta esto como una llamada al método

Grid.SetRow al que se le pasa como parámetros el elemento dentro del cual se referencia a la propiedad (en este caso el botón) y el valor asignado a la propiedad ("1").

Esta forma de referirse a algunas propiedades es una facilidad sintáctica que ofrece *XAML* y que simplifica la notación y ayuda a ganar en flexibilidad. Es muy utilizada en elementos que distribuyen otros elementos, como los paneles por ejemplo (Katrib, Del Valle, Sierra, & Hernández, 2007).

4.3.4 Convertidores de Tipo

En *XAML* los valores de los atributos se expresan siempre como un *string,* pero no siempre los valores con los que realmente se corresponden en *.Net* son de tipo string. Para esto, *XAML* se apoya en el mecanismo de convertidores de tipo. Como su nombre indica, permiten convertir valores de un tipo a otro, generalmente valores de tipo string a valores de algún otro tipo propio de *.Net*.

Por ejemplo en la figura 3.7 se asigna el string "1" a la propiedad *Row* del elemento *Grid*, cuando realmente en el elemento *.Net Grid* esta propiedad es de tipo entero. El convertidor de tipo transforma la cadena "1" en el número entero 1 y es éste valor el que se le asigna a la propiedad (Katrib, Del Valle, Sierra, & Hernández, 2007).

4.3.5 Extensiones de marcado

La combinación de convertidores de tipo y las propiedades posibilita la inicialización de la mayoría de éstas. Hasta ahora se ha visto que lo que se asigna a una propiedad en *XAML* es una cadena (que podrá ser convertida al objeto correspondiente). Esto da como resultado valores constantes o estructuras fijas, es decir, valores conocidos en tiempo de diseño y compilación. Sin embargo, sería deseable que *XAML* ofreciera mayor flexibilidad a modo de poder asociar a una propiedad un valor que no se conozca en tiempo de diseño y compilación. Por ejemplo asociar a una propiedad un valor que dependa de otra propiedad y que pueda variar en tiempo de ejecución. Para esto *XAML* propone lo que se conoce como extensiones de marcado *(markup extensions)*.

El compilador *XAML* reconoce que se está usando una extensión de marcado cuando la cadena que contiene el valor a asignar a una propiedad está encerrada entre llaves { }.

Un ejemplo de extensión de marcado muy utilizada es *Binding,* la cual se utiliza para hacer enlace con datos (Katrib, Del Valle, Sierra, & Hernández, 2007):

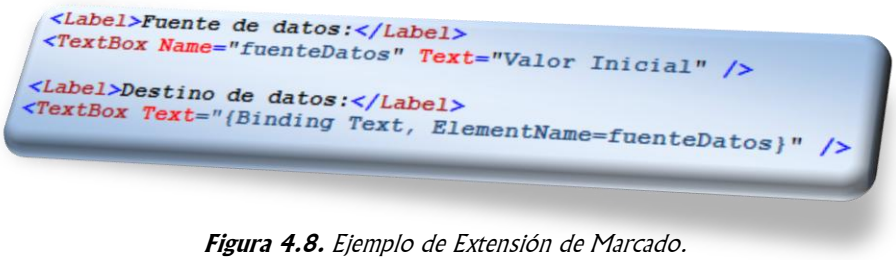


Figura 4.8. *Ejemplo de Extensión de Marcado.*

4.3.6 El code-behind

Si se hiciera clic en el botón de la aplicación de la figura 3.7, no ocurriría nada, puesto que no se ha definido ningún comportamiento asociado a esta acción.

XAML da soporte al concepto *code-behind*. Todo archivo *XAML* tiene un correspondiente archivo con código ejecutable. La idea es que el *XAML* de la apariencia y estructura de la interfaz de usuario, mientras que el archivo *code-behind* debe dar el comportamiento, relacionado con la lógica de negocio (Katrib, Del Valle, Sierra, & Hernández, 2007).

4.3.7 XAML y SilverLight

A pesar de que *XAML* fue desarrollado para el uso en la plataforma Windows, actualmente es posible utilizarlo en otras plataformas gracias a *SilverLight*.

Silverlight (antes conocido como *Windows Presentation Foundation/Everywhere, WPF/E)* es una nueva tecnología de ***presentación web*** creada para su ejecución en ***distintas plataformas***. Hace posible un uso más completo y atractivo de los recursos visuales e interactivos, pudiéndose ejecutar en todos los entornos, en múltiples dispositivos y sistemas operativos de escritorio (como por ejemplo en *Macintosh* de *Apple)* (Moroney, L., 2007).

XAML constituye la base de la capacidad de presentación de *Silverlight*.

Con todo lo expresado en este capítulo se da por finalizado la ilustración de qué, cómo es y cuáles son los objetivos de *XAML*.

4.4 RESUMEN

La aparición del *Framework 3.0* trae consigo cuatro nuevas tecnologías que se apoyan sobre la versión anterior del *Framework de .Net*, el *2.0*, como son: *WPF (Windows Presentation Foundation), WCF (Windows Communication Foundation), WF (Windows Workflow Foundation) y CardSpace* o también conocido como *InfoCard*.

Figura 4.9. *Nuevas tecnologías del .Net Framework 3.0.*

Pero no sólo estas tecnologías son las novedades que aporta el *Framework 3.0 de .Net*, también surge con él un nuevo lenguaje, *XAML*.

Para el punto de vista del presente proyecto la tecnología más importante que aporta el nuevo *Framework* es *WPF* (*Windows Presentation Foundation*), la cual aglutina una gran cantidad de capacidades para desarrollar interfaces de usuario (*IU*) muy ricas y con múltiples características. Antes de su aparición, eran necesarias varias tecnologías para crear *IU* con las características que ahora pueden realizarse mediante tan sólo la plataforma unificada *WPF*.

XAML es un lenguaje de marcado declarativo, basado en *XML*, y pensado para desarrollar interfaces de usuario en *WPF*. Su principal característica es la capacidad que ofrece para separar la ***descripción gráfica*** de la interfaz de su ***lógica de negocio***, expresando la definición de la *IU* mediante *XAML* y la lógica a través de *C#* o *VB .Net*. Gracias a esta capacidad los diseñadores y los desarrolladores pueden trabajar juntos sin tener que sufrir todos los inconvenientes que anteriormente surgían como fruto de su colaboración. Este nuevo lenguaje está revolucionando el mundo del desarrollo software.

5. AUTOMATIZACIÓN DE PRUEBAS EN INTERFACES DE USUARIO GRÁFICAS

Ya que la aparición de la nueva tecnología *WPF* junto con el lenguaje declarativo que trae consigo, el *XAML,* han sido ideados para permitir a los desarrolladores crear *interfaces de usuario gráficas* muy ricas y potentes, la realización de un buen testeo se convierte en condición obligatoria y necesaria para poder ofrecer un producto software de calidad y que éste no se límite tan sólo a la espectacularidad de su diseño sino también a la correcta ejecución de su funcionalidad y que dicha ejecución se lleve a cabo desde su interfaz de un modo tan eficiente como eficaz.

Estas premisas son de primordial cumplimiento para los Sistemas de Información Corporativos pues las decisiones de sus usuarios pueden variar en función de la información que éstos muestren como salida. En definitiva los Sistemas de Información Corporativos han de ser herramientas software de calidad.

Los managers de proyectos software y los desarrolladores de las aplicaciones de hoy en día se enfrentan al desafío de crear software de calidad con constantes disminuciones de calendario y con los recursos mínimos. Para asegurarse que el producto desarrollado es de calidad han de testear adecuadamente la herramienta, pero lo han de hacer de la manera más rápida y completa posible y para lograr esta meta, las organizaciones recurren a la automatización de pruebas.

5.1 CALIDAD DEL SOFTWARE

La *calidad de software* ha sido usada utilizada en ocasiones como un simple argumento de venta, algo publicitario, mientras que en otras se han realizado verdaderos estudios formales y usos de métricas para el desarrollo de software.

Desde el lado de la *Ingeniería del Software* la *calidad del software* es algo muy complicado de definir y de enmarcar en un simple concepto teórico. Aún así se muestran dos definiciones para intentar arrojar algo de luz sobre el concepto que se está tratando:

"*La calidad del software es el grado con el que un sistema, componente o proceso cumple los requisitos especificados y las necesidades o expectativas del cliente o usuario.*" (IEEE, Std. 610-1990).

"Concordancia del software producido con los requisitos explícitamente establecidos, con los estándares de desarrollo prefijados y con los requisitos implícitos no establecidos formalmente, que desea el usuario." (Pressman, 1998).

Dado que sólo con definiciones es muy complejo hacerse una idea de lo que la que realmente es la *calidad del software* se exponen a continuación las diversas características que permiten describirla y los elementos que importan específicamente al diseñador de software a fin de conseguir un software de calidad, destacando las más importantes.

5.1.1 Factores que determinan la calidad

Se centran en tres aspecto importantes de un producto software (Buades, 2002):

- ***Características operativas***:
  - *Corrección*. Esta característica se puede extraer de las dos definiciones mostradas, indica si el software hace lo que el cliente quiere, en definitiva si cumple con los requisitos.
  - *Fiabilidad*. El software ha de hacer lo que se quiere que haga pero de una manera fiable todo el tiempo.
  - *Eficiencia*. Tiene que ver con el uso eficiente de los recursos que necesita un sistema para su funcionamiento.
  - *Seguridad.* Incluye varias características además de la seguridad, como la integridad, control de fallos, etc.
  - *Usabilidad*. El software ha sido diseñado para ser utilizado sin un gran esfuerzo por los usuarios para los que fue diseñado.

- ***Capacidad de soportar los cambios***:
  - *Mantenibilidad*. Ha de ser implementado de manera que se facilita el mantenimiento así como también la corrección de los errores que se vayan detectando.
  - *Flexibilidad*. El software debe ser diseñado de tal manera, que permita ajustarlo a los cambios de requisitos del cliente. Esta característica es crucial, debido al inevitable cambio del contexto en el que se desempeña un software.

- ***Adaptabilidad a nuevos entornos***:
  - *Portabilidad*. Esta característica responde a la pregunta de si se el software podrá ser utilizado en otra máquina, en otros entornos, etc.
  - *Reusabilidad*. Si alguna parte vuelve a ser necesaria para otro desarrollo ha de poder ser reutilizable.
  - *Interoperabilidad*. Esta característica indica si el software podrá interactuar con otros sistemas.

Como puede observarse, las diversas características con las que se desea que cumpla un software de calidad varían ampliamente. Algunas tienen que ver con el usuario que interactúa con el sistema, otras con el líder de proyecto y diseñadores, otras características parecen muy abstractas y hasta indefinidas, etc. Para ordenar este aparente caos de indefiniciones y características abstractas, con el fin de poder medirlas, estimarlas e implementarlas, la *Ingeniería del* Software ha desarrollado desde los primeros días de su existencia, diferentes procesos de desarrollo.

Esta búsqueda para poder controlar y medir la *calidad del software*, es tal vez una de las principales causas que han inspirado el estudio y definición de un sinfín de metodologías, técnicas y herramientas de la *Ingeniería del Software*.

Hay una gran variedad de procesos de donde tomar los elementos más convenientes para alinear los desarrollos con algunas características de la *calidad del software* vistas previamente, estos son algunos elementos:

- Se necesita coherencia desde el principio del proyecto. En este momento deben definirse, cuantificarse y/o especificarse las características de calidad a cumplirse en el producto.
- Se requieren las herramientas necesarias que ayuden al equipo para llevar adelante todas las tareas necesarias en relación a alcanzar los objetivos de calidad planteados.
- Es muy importante también, disponer de personas preparadas técnicamente y liderados por al menos un profesional con experiencia, que formen un equipo con la capacidad de adaptarse y mejorar continuamente.

Como puede observarse, elegir el camino del desarrollo de *software de calidad* no significa disponer de grandes inversiones, sino de alinear los recursos disponibles, prepararlos y coordinarlos adecuadamente. Llegado el momento de escalar, o desear el logro de alguna certificación para ampliar mercados, o sencillamente buscar ser una empresa que logre desarrollar productos de calidad, será mucho mejor y más simple, siempre que se sigan algunos elementos mínimos de *buenas prácticas en el desarrollo de software*.

Para ofrecer a los clientes un *software de calidad* es condición necesaria realizar una implementación libre de *errores* o con el menor número posible (Kaner, Falk, & Nguyen, 1993). Pero, ¿qué es exactamente un *error software* también conocido como ***bug***?

Un error de software es una incorrección o anomalía presente en el código de un programa informático, lo cual provoca que el programa no haga lo que su usuario final espera razonablemente que haga (Kaner, Falk, & Nguyen, 1993).

La medida en que un programa tiene errores se mide por el grado en que no sea útil (la aplicación no hace lo que el usuario quiere: software de baja calidad).

A continuación se muestran algunas de las subcategorías en los *errores de interfaz de usuario,* que son los que especialmente más interesan en este proyecto.

- ***Funcionalidad:*** Cualquier interfaz de usuario tiene un problema de funcionalidad si no hace lo que debe o lo hace torpe o incompletamente.
- ***Comunicación:*** La manera en la que el programa se comunica con el usuario debe ser la correcta. Ha de comunicarse en su lenguaje, mostrar en pantalla la información suficiente y de una manera comprensible para el usuario de modo que pueda saber en cada momento qué es lo que está haciendo la aplicación. La ayuda que se ofrezca ha de ser útil y estar bien presentada. Cuando no se cumple alguna de estas condiciones se presenta un problema para el usuario y por tanto un *bug*.
- ***Estructura de comandos:*** Este tipo de errores indican que es fácil perderse por la interfaz del programa, que existen comandos confusos o fáciles de equivocar con otros.
- ***Comandos olvidados:*** Toda interfaz que obliga al usuario a pensar de una manera rígida, antinatural o ineficiente, que no puede ser customizardo para adaptarlo a su estilo de trabajo o necesidades, siendo un programa en el que la customización en una cuestión importante, presenta un error de *comandos olvidados*.
- ***Rendimiento:*** La velocidad es la esencia del software interactivo. Cualquier cosa que pueda hacer sentir al usuario que el programa está trabajando lento es un problema.
- ***Salida:*** A menudo los programas muestran, imprimen o guardan información. El usuario ha de obtener los resultados que quiere en la salida. Las impresiones han de tener sentido, se deben leer los gráficos, es posible que los datos almacenados tengan que ser leídos por otra aplicación, se han de guardar con el formato correcto. La salida puede ser reorientada a su elección: terminal, impresora, archivo, etc.

5.2 AUTOMATIZACIÓN DE PRUEBAS SOFTWARE

A primera vista, puede parecer bastante sencillo automatizar el testeo de una aplicación: adquirir alguna de las populares herramientas de testeo, grabar manualmente los casos de test y ejecutar la secuencia almacenada cuando se quiera. Desafortunadamente, no es tan simple ya que con sólo esos pasos no se conseguiría jamás una buen testeo de la aplicación.

Al igual que es necesario muchas más cosas para diseñar un software que conocer tan sólo un lenguaje de programación, para automatizar pruebas hace falta mucho más que conocer una herramienta de testeo (Fewster, 1999).

El software ha de ser probado para tener la seguridad que funciona como debe. Dicho testeo debe ser efectivo en la búsqueda de cualquier defecto que tenga, pero también debe ser eficaz realizando las pruebas de la manera más rápida y barata posible y es por esto último por lo que se utilizan las pruebas automatizadas.

La *automatización de pruebas software* puede reducir significativamente el esfuerzo requerido para un testeo adecuado o puede incrementar elocuentemente las pruebas que se pueden realizar con un tiempo y recursos limitados. Las pruebas automáticas pueden ejecutar en minutos lo que llevaría horas hacerlo manualmente. En determinadas circunstancias se ha logrado ahorrar el 80% del esfuerzo de testeo manual. Hay ocasiones en las que las organizaciones no ahorran dinero o esfuerzo directamente, pero a cambio sus pruebas automáticas les han permitido producir software de *mayor calidad* más rápido de lo que lo hubieran hecho realizando únicamente un testeo manual.

Un plan de test de *pruebas automatizadas* que ha sido bien elaborado y se ha ido mejorando en el tiempo puede permitir, con sólo pulsar un botón, que las pruebas se ejecuten durante la noche cuando las máquinas están ociosas y sin que sea necesario que haya nadie pendiente de su marcha (Fewster, 1999).

Los *test automáticos* son repetibles, utilizando exactamente las mismas entradas y en la misma secuencia ejecución tras ejecución, algo que no se puede garantizar con el testeo manual. Asimismo el testeo automático posibilita que incluso el más pequeño cambio pueda quedar completamente probado con el mínimo esfuerzo.

5.3 AUTOMATIZACIÓN DE PRUEBAS EN INTERFACES DE USUARIO GRÁFICAS

Las aplicaciones informáticas van evolucionando y son cada vez más sofisticadas. Lo mismo ocurre con sus interfaces gráficas, las cuales proporcionan una mayor facilidad de uso para el cliente, y un mayor beneficio para el vendedor. Como cualquier otro software, el código de la *IU* debe ser testado también. Esta capa del software es la más cercana al usuario y la que presenta mayor complejidad a la hora de establecer una automatización sobre las pruebas.

Dicha complejidad radica en la propia naturaleza de las *interfaces usuario gráficas* y en la forma en que se desarrollan y mantienen lo que implica que sus pruebas sean diferentes de otras formas de test automatizadas (Fewster, 1999):

- La *GUI* tiende a cambiar con más frecuencia que la funcionalidad que invoca – a medida que se van añadiendo nuevas características a la aplicación la interfaz se reorganiza para mostrar los nuevos datos de maneta coherente. Esto significa que el mantenimiento cobra mayor importancia dado que las nuevas versiones probablemente dejen obsoletas las pruebas *GUI* existentes.
- El código de la *interfaz* es complejo - siempre existen múltiples formas de llevar a cabo una determinada operación, la cual implica más elementos a testar.
- Las herramientas para testar interfaces de usuario son complejas en sí mismas. La mayoría de ellas tienen su propio lenguaje de script que es tan robusto y difícil de aprender como el *BASIC* o C.
- Se ejecutan como un proceso separado de los procesos de la aplicación que se está testando lo que significa que es necesario sincronizar la velocidad de ejecución del script con la velocidad de la aplicación que se prueba, un trabajo complicado que requiere técnicas diferentes para situaciones distintas.
- Las herramientas de testeo de *IU* deben ser capaces de manipular y consultar un conjunto de controles comunes tales como *Botones, ListBox, TextBox, Menús,* etc. lo que trae como consecuencia un gran número de funciones a exportar para el usuario encargado de escribir el script de test.
- Las herramientas de testeo de *interfaces de usuario gráficas* modernas proporcionan medios para manejar los controles de ventanas más comunes de una forma abstracta, utilizando semánticas relacionadas con la función del control (como por ejemplo ***button_press ("OK");***), pero las funciones que manejan ***controles personalizados*** son muy primitivas y con frecuencia requieren el empleo de las coordenadas X e Y de la pantalla. Puesto que la mayoría de las aplicaciones utilizan cierto número de

controles personalizados, el trabajo del usuario que debe escribir el script de prueba se hace más difícil a medida que incrementa el número de *controles personalizados*.

5.3.1 Requisitos de los Test Automatizados para Interfaces de Usuario Gráficas

La creación de un conjunto de pruebas (también conocido como *Batería de Pruebas*, *Test Suite o Test Plan*) para una interfaz de usuario, supone un proyecto de ingeniería software que requiere las mismas habilidades y disciplina que implementar la propia aplicación testada. Dado que la *Batería* se utiliza sobre numerosas versiones del producto debe satisfacer los siguientes requisitos básicos:

- Debe ser *mantenible*, puesto que la siguiente versión de la aplicación que se está testando convertirá algunos de los casos de test en obsoletos, requerirá la modificación de otros y demandará la creación de nuevos casos de test para las funcionalidades añadidas.
- Debe ser modular, ya que se tiene que evitar la situación en que el cambio de alguna parte la *Batería* provoque que las pruebas comiencen a fallar en otras áreas.
- Debe ser robusta, porque se debe impedir que cualquier pequeño cambio en la aplicación exija grandes cambios en la *Batería*.
- Debe estar bien documentada ya que en la mayoría de los casos el ingeniero que hizo la Batería no será el mismo que la actualice para la nueva versión.
- Y en la medida de lo posible debe estar construida con componentes reutilizables para que aquellos que lleguen después no tengan que *"reinventar la rueda"*.

Para poder realizar un buen testeo de la interfaz y rebajar un poco la complejidad de este proceso es muy importante que la lógica de negocio y el acceso a datos esté apropiadamente separado de la *interfaz de usuario* ya que, de este modo, se dota a la aplicación de una mayor independencia respecto a la plataforma en la que deba visualizarla el usuario, y puede obtenerse mayor cobertura de código durante las pruebas de ambas partes.

5.4 HERRAMIENTAS PARA LA AUTOMATIZACIÓN DE PRUEBAS GUI

Existen diferentes opciones para llevar a cabo baterías de pruebas sobre la *IU*, que dependen del tipo de aplicación que se vaya a testear (de consola, con interfaz gráfica, web, etc.) y del lenguaje o plataforma de desarrollo. En el caso de las pruebas sobre la *IU* en aplicaciones web, existen frameworks de prueba como Selenium, etc. pero el presente proyecto se va a centrar en las herramientas que permiten testar *interfaces de usuario gráficas*.

5.4.1 Las herramientas de tipo capture/replay

Algunos fabricantes de herramientas de testeo automático para *interfaces gráficas* promocionan una metodología de guardado/reproducción (también conocido como *capture/replay*) como el centro de diseño para emplear sus herramientas. Construir tests utilizando este método implica llamar a la herramienta de test en modo "*capture*" y posteriormente manipular la aplicación a testar realizando una serie de acciones útiles. La herramienta de test captura las acciones y las graba en un script. Las cuales pueden después ser repetidas, para accionar la aplicación de forma automática.

El uso de *capture/replay* ha funcionado bastante bien con las interfaces de línea de comandos (*CLI*), pero existen problemas importantes cuando se trata de aplicar en una *GUI*. El problema más evidente que se encuentra es que la pantalla en un sistema *GUI* puede ser diferente, mientras que el estado interno de la aplicación es el mismo, haciendo la validación automática extremadamente difícil. Esto es debido a que la *interfaz gráfica* permite que los objetos varíen en apariencia y colocación en la pantalla, las fuentes pueden ser diferentes, ventana de colores o tamaños pueden modificarse también, pero la salida del sistema es básicamente la misma. Todo esto sería evidente para un usuario, pero no lo es para un sistema de validación automatizada.

Para combatir este y otros problemas, los desarrolladores de este tipo de herramientas recogieron los datos de la interacción entre la *GUI* y el sistema de ventanas subyacente. Al capturar los eventos de ventana y registrar las interacciones con el sistema pudieron elaborar un esquema que era independiente de la apariencia de la *GUI*. A partir de este punto tan sólo se captura la sucesión de eventos, pero siendo necesario realizar algún filtrado, ya que de otra manera la captura de éstos sería muy detallada y la mayoría de ellos no serían pertinentemente directos al problema. Este enfoque resulta más fácil de aplicar si se utiliza una arquitectura MVC, haciéndose la vista (la *GUI*) lo más sencilla posible, mientras que el modelo y el controlador deben soportar toda la lógica. Otro enfoque es utilizar una interfaz HTML o una arquitectura de tres niveles que posibilita también una buena separación entre la interfaz de usuario y el resto de la aplicación.

Los fabricantes de este tipo de herramientas promueven la metodología *capture/replay* como una forma de bajo coste para generar gran cantidad de código test. Pero, además de los problemas comentados, el sistema *capture/replay* no es tan económico como se apunta puesto que quizá posibilite que personas sin experiencia en codificación produzcan gran cantidad de código de test, pero ese código no tiene ninguno de los requisitos apropiados, que han sido expuestos antes, para los *test automatizados* de las aplicaciones que presentan *interfaces de usuario gráficas*.

Otra forma de ejecutar pruebas automatizadas sobre una *GUI* es incorporar un driver en la propia *interfaz gráfica* de manera que los comandos o eventos pueden ser enviados al software desde otro programa. Este método de enviar eventos directamente y recibirlos en la *GUI* es muy conveniente para realizar los test automáticos ya que la entrada y la salida de la prueba puede ser totalmente automatizada y las probabilidades de error por parte del usuario se eliminan por completo.

5.4.2 EJEMPLOS DE HERRAMIENTAS DE AUTOMATIZACIÓN GUI

5.4.2.1 QuickTest Professional de Mercury

Quick Test Professional (*QTP*) es una herramienta para el testeo de *Interfaces de Usuario Gráficas* creada por Mercury (adquirida por *HP Software* en Diciembre de 2006) que permite automatizar las acciones que pueda realizar un usuario. Las principales utilidades de este producto son las de automatizar pruebas *GUI*, de regresión[2] y funcionales[3]. *QTP* utiliza un lenguaje de script, construido en base a VBScript, para especificar el código de la prueba y para manipular los objetos y controles de la aplicación que se está testeando.

El funcionamiento de *QTP* se basa en la identificación de los objetos en la *interfaz gráfica* de la aplicación y realiza las operaciones deseadas sobre ellos (como por ejemplo click's de ratón o eventos de teclado). También puede ser utilizado como espía de la interfaz gráfica, es decir para capturar las propiedades de los objetos tales como el nombre, el identificador, etc.

Algunas de las características principales de QuickTest Professional son (Wikipedia - HP QuickTest Professional):

- Habitualmente el desarrollo inicial para automatizar pruebas con *QTP* es realizado mediante *capture/replay.* Las acciones del usuario son capturadas a través de un objeto COM. Estas acciones son traducidas y almacenadas como comandos VBScript. Una vez las acciones son almacenadas en un script, éste puede ser editado. Tras pulsar el botón de reproducir script (*replay*) todas esas acciones son ejecutadas de nuevo automáticamente. Para ello *QTP* identifica los objetos y realiza las mismas operaciones, utilizando de nuevo el mismo objeto COM.
- Ofrece múltiples plug-ins para permitir al usuario hacer la mejor grabación del script posible. Hay plug-ins web para automatizar casos de test correspondientes a aplicaciones web. Existen también plug-ins para controles ActiveX, objetos VB y para objetos .NET.

[2] Se denomina ***pruebas de regresión*** a cualquier tipo de pruebas software que intentan descubrir nuevos errores (*bugs*), carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, provocados por cambios recientemente realizados en la aplicación y que anteriormente a las modificaciones no era propensa a este tipo de error. Esto implica que el error tratado se reproduce como consecuencia inesperada del citado cambio en el programa.

[3] Una ***prueba funcional*** es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Las pruebas funcionales se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático.

- *Recovery,* es el nombre para el control de excepciones de *QTP,* y tiene el objetivo de poder continuar con el test si se produce un error en la aplicación. Por ejemplo si se produce una excepción *QTP* muestra un mensaje e indica al usuario de que manera ha de reiniciar la aplicación y continuar con el resto de la prueba.
- *QuickTest Pro* genera el resultado de las pruebas en un fichero con formato XML. Dicho fichero contiene detalles como test *pasados,* test *fallados,* mensajes de error e información de ayuda que permite al usuario descubrir la causa de los errores.

5.4.2.2 Rational Robot de IBM

Rational Robot permite a los equipos de pruebas automatizar las pruebas de regresión y funcionales de aplicaciones .NET, Java, Web y otras aplicaciones basadas en interfaces de usuario gráficas.

Algunas de las características principales de *Rational Robot* son (IBM - Rational Robot - http://www-01.ibm.com/software/awdtools/tester/robot):

- Es una herramienta versátil de pruebas de configuración, regresión y funcionales para entornos en los que se desarrollan las aplicaciones utilizado más de un IDE y/o lenguaje de programación.
- Da soporte a múltiples tecnologías de interfaz de usuario gráficas para cualquier entorno: desde Java y la Web hasta todos los controles de VS.NET, incluidos VB.NET, J#, C# y C++ gestionado.
- Proporciona casos de prueba para objetos comunes, como menús, listas y mapas de bits y casos de prueba especializados para los objetos específicos del entorno de desarrollo.
- Simplifica la configuración de los test. Permite distribuir test funcionales entre varias máquinas con facilidad, cada uno con una configuración distinta. El mismo test funcional puede ser ejecutado simultáneamente, reduciendo el tiempo para identificar problemas con alguna configuración específica.
- Posibilita la reutilización de scripts. *Rational Robot* permite ejecutar el mismo script de prueba en una aplicación ejecutada en *Microsoft Windows XP, Windows ME, Windows 2003, Windows 2000, Windows 98 ó Windows NT.*
- *Rational Robot* genera los scripts de test en *SQABasic,* un lenguaje de script integrado que permite ver y modificar los test mientras son grabados.

5.4.2.3 SilkTest de Segue Software

SilkTest® es una herramienta para la automatización de pruebas de regresión y funcionales creada por Segue Software®, la cual fue comprada por Borland® en 2006. Posee una interfaz gráfica intuitiva con capacidad para realizar pruebas del tipo *capture/replay*.

Algunas de las características principales de SilkTest® son (Borland Software - http://www.borland.com):

- Presenta un uso bastante sencillo y permite la automatización de pruebas funcionales y de regresión que cubre gran cantidad de entornos de desarrollo y tecnologías sin costes añadidos como plug-ins, conectores, etc.
- Un asistente ayuda al usuario inexpertos a desarrollar test útiles rápidamente.
- Los test, robustos y resistentes, son soportados por un lenguaje propio, llamado 4Test, flexible, orientado a objetos, de cuarta generación y expresamente creado para la creación de script de pruebas automáticas.
- Posee un método de recuperación de errores que devuelve el sistema al momento anterior al fallo y continúa con la ejecución del test.
- Da soporte a las tecnologías .NET, Java, DOM, así como también aplicaciones Flex que utilizan Internet Explorer, Firefox, Flex Player y Adobe AIR.
- El modo utilizado por SilkTest® para reconocer cada control o ventana de la aplicación es asignar un identificador a cada uno y relacionarlos con una serie de propiedades de éste que lo identifiquen. Además posee una tecnología, llamada Instanciado Dinámico, que permite referirse a objetos que se generan dinámicamente.

5.5 RESUMEN

Los Sistemas de Información Corporativos demandan ***herramientas de calidad*** puesto que las importantes decisiones que deban tomar sus usuarios pueden variar en función de la información que genere el software como respuesta a las entradas recibidas.

La ***calidad software*** se puede resumir de una manera muy escueta como *el grado con el que un sistema cumple los requisitos especificados y las expectativas del cliente,* es decir la aplicación hace lo que el usuario espera que haga.

El desarrollo de ***software de calidad*** no se basa tan sólo en una implementación cuidadosa de la herramienta de modo que vaya cumpliendo con exactitud las restricciones del cliente sino también en la elaboración de un completo plan de test que asegure que lo anterior se está cumpliendo.

El problema que surge entonces es que tanto el desarrollo como el testeo deben realizarse en un espacio de tiempo y con unos recursos demasiado limitados. Es por esto que las organizaciones optan por automatizar las pruebas de modo que les permita ahorrar gran cantidad de tiempo y recursos o que con las mismas restricciones sean capaces de realizar testeos de las aplicaciones mucho más completos, posibilitando además producir software de *mayor calidad* más rápido de lo que lo hubieran hecho realizando únicamente un testeo manual.

Pero esta automatización no es tan simple como puede parecer a priori, realizar una correcta automatización de pruebas implica mucho más que conocer una herramienta de testeo. La creación de una *Batería de Pruebas* para una interfaz de usuario, supone un proyecto de ingeniería software que requiere las mismas habilidades y disciplina que implementar la propia aplicación testada.

Al igual que las propias aplicaciones informáticas sus interfaces gráficas son cada vez más sofisticadas y como cualquier otro software, su código también debe ser testado. La *IU* es la capa del software más cercana al usuario y la que presenta mayor complejidad a la hora de establecer una automatización sobre sus pruebas debido a la forma en que se desarrollan y mantienen, pues es la parte de la aplicación que más modificación recibe.

Para realizar un buen plan de test de pruebas automatizadas sobre una GUI es necesario que la batería cumpla con algunos requisitos, como son: debe ser mantenible, modular, robusta, reutilizable y ha de estar bien documentada.

En las aplicaciones para la automatización de test de interfaces de usuario gráficas es muy común el tipo de herramientas capture/replay. Que consisten activar la herramienta de test en modo "*capture*" y manipular la aplicación a testar realizando una serie de acciones. Las acciones son capturadas y

almacenadas en un script. Las cuales pueden después ser repetidas, para accionar la aplicación de forma automática.

Este tipo de herramientas permiten que usuarios sin experiencia en codificación generen gran cantidad de código test pero presentan un problema muy evidente, la pantalla del sistema *GUI* puede ser diferente, mientras que el estado interno de la aplicación es el mismo. Esto sería algo muy evidente para un usuario, pero no lo es para un sistema de validación automatizada.

La manera en que esto se intenta solucionar es capturando únicamente la sucesión de eventos que tienen lugar entre la interacción de la GIU y el sistema de ventanas subyacente. De este modo se puede elaborar un esquema independiente de la apariencia de la *GUI*.

Otra posible manera de automatizar las pruebas sobre una *GUI* es la de incorporar un driver en la propia *interfaz gráfica* y que sea éste el encargado de recibir directamente los eventos que son enviados desde otro programa. Esta manera de realizar los test automáticos hace que la entrada y la salida de la prueba puedan ser totalmente automatizadas y que las probabilidades de error por parte del usuario se eliminen por completo.

6. CONCLUSIONES AL ESTADO DEL ARTE

En los tres capítulos anteriores se han descrito los pilares fundamentales en los que se basa este proyecto y que son:

- Los Sistemas de Información Corporativos en las organizaciones.
- *XAML*, el nuevo lenguaje para el desarrollo de interfaces de usuario gráficas en *WPF*.
- La automatización de pruebas en interfaces gráficas y como pueden ayudar para realizar desarrollos software de calidad.

En este capítulo se establecerán las relaciones existentes entre estas tres áreas y serán las que permitan de manera efectiva construir la base sobre la que desarrollar el resto del proyecto.

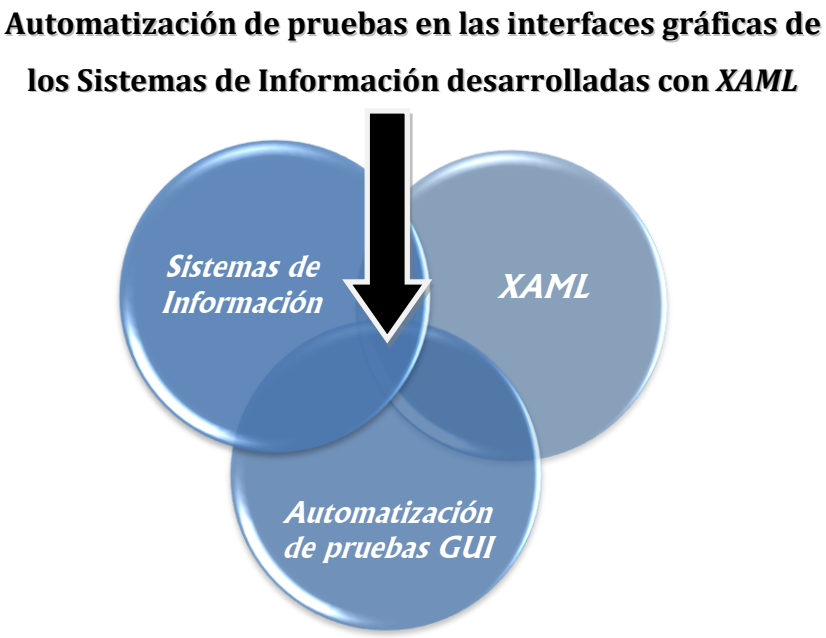


Figura 6.1. *Relación entre los SI, XAML y la automatización de las pruebas GUI.*

El estándar *XAML* representa para los *Sistemas de Información de las empresas* una oportunidad para desarrollar mejores interfaces, es decir, interfaces más eficaces, más atractivas, capaces de ofrecer, a través del binomio *XAML-WPF*, gran cantidad de características o técnicas, como gráficos 2D y 3D, animaciones, características tipo web, etc., interfaces que presenten una amplia riqueza visual e interactiva. Además la creación de estas interfaces supone un menor esfuerzo en cuanto a tiempo de desarrollo se refiere.

Debido a esta nueva posibilidad que ofrece *XAML* para crear *interfaces de usuario* muy ricas y complejas, tanto en aspectos visuales como interactivos, y para poder ofrecer el software de calidad que los *Sistemas de Información* requieren se convierte en obligatorio la realización de un buen y completo testeo sobre la aplicación, abarcando también las nuevas y potentes *interfaces de usuario gráficas*.

Dicho testeo debe asegurar que el software funciona y se comporta como debe, pero no debe hacerlo tan sólo de una manera efectiva sino que además tiene que ser eficaz, siendo las pruebas automatizadas una opción para alcanzar este último objetivo.

Las *pruebas automatizadas* se muestran como una alternativa para que las organizaciones consigan ahorrar gran cantidad de tiempo y recursos produciendo software de mayor calidad más rápido de lo que lo harían utilizando tan sólo el testeo manual.

Esta circunstancia habilita al presente *Proyecto Final de Carrera* como una de las iniciativas más punteras y prometedoras en el desarrollo de ***herramientas para el testeo automatizado de las interfaces gráficas WPF/XAML*** facilitando la integración de los *SI* con las nuevas tendencias para el desarrollo de interfaces humanas.

Esta iniciativa surge a partir de un proyecto en el cual, la *Universidad Carlos III de Madrid* colabora con una empresa líder en el desarrollo y comercialización de software paquetizado de ámbito empresarial. El objetivo de este proyecto de colaboración es el desarrollo de una ***herramienta, denominada X-Aute, capaz de automatizar el testeo de las novedosas interfaces de usuario gráficas*** que han sido creadas, para la nueva generación de los productos software desarrollados por dicha empresa, a través de la integración de las tecnologías de interfaz de usuario más avanzadas, como son las tecnologías basadas en ***.Net,*** el sistema de interfaces gráficas ***Windows Presentation Foundation® (WPF) y XAML***.

La aplicabilidad de los estudios y los análisis realizados, en el presente *Proyecto Final de Carrera*, en el entorno productivo empresarial está garantizado por la adopción del estándar *XAML* en las aplicaciones de *Sistemas de Información comerciales* que la empresa líder, con la que colabora la Universidad Carlos III de Madrid, lleva a cabo y que se pretende sean comercializas y por la aplicación, en el entorno de desarrollo de la empresa citada, del producto final de este

Proyecto Fin de Carrera. Asimismo tales estudios, análisis e implementación realizados asentarán las bases para posteriores desarrollos relacionados con la automatización de pruebas sobre interfaces *WPF*.

En resumen *X-Aute* nade de la conjunción de tres áreas de conocimiento que de por sí son campos prometedores tanto en la investigación aplicada como en el ámbito empresarial. *X-Aute* aprovecha lo mejor de los tres mundos para aportar una solución innovadora y aplicable a las organizaciones del futuro.

SECCIÓN
III
DESCRIPCIÓN DE LA SOLUCIÓN

7. FUNCIONALIDADES

El estudio preliminar realizado *(véase la Sección II)* ha permitido identificar las características que ha de poseer una herramienta de testeo automatizado de interfaces de usuario gráficas y más concretamente de interfaces *WPF.* Después de analizar dichas características se ha identificado la necesidad de construir una herramienta como *X-Aute*, una aplicación para el testeo automatizado de las *interfaces de usuario gráficas* desarrolladas mediante *WPF*.

En primer lugar se ha de comentar cuál es el objetivo fundamental de *X-Aute.* Come ya se ha señalado varias veces a lo largo de este *Proyecto Fin de Carrera* la herramienta desarrollada debe ofrecer las funcionalidades necesarias para poder realizar pruebas *GUI* sobre las novedosas interfaces creadas con el *.Net Framework 3.0* de Microsoft fundamentándose principalmente en la nueva tecnología que este trae consigo, *WPF*, así como también en el nuevo lenguaje descriptivo de interfaces gráficas que lo acompaña, *XAML*. Pero no se trata únicamente de una herramienta que permita realizar una serie de test de cualquier modo sino que permita automatizar totalmente tanto la entrada como la salida de la prueba consiguiendo así eliminar la interacción del humano y por tanto eliminar por completo las probabilidades de error por parte de éste. Al mismo tiempo que posibilita la repetición del test en el momento deseado con exactamente las mismas características.

Para poder realizar dicha automatización la primera condición que se plantea de necesario cumplimiento es que *X-Aute,* permita crear los scripts de prueba que serán almacenados y más tarde podrán ser ejecutados automáticamente en el momento que convenga.

X-Aute hace que la creación de los scripts de test se realice de un modo muy sencillo e intuitivo. Para escribir estos test se utilizará el lenguaje de programación *C#*. Sin embargo, el usuario encargado de dicha tarea no debe poseer grandes conocimientos es este ni en ningún otro lenguaje, ni si quiera en el paradigma de la programación orientada a objetos, pues la herramienta facilita el trabajo del usuario realizando una gran cantidad de operaciones que para él serán transparentes e inicializando los objetos a los que el usuario únicamente se encargará de invocar indicando que acción quiere que realice cada control. Por si no fuera suficiente, se entrega una amplia documentación acerca del uso de la herramienta, el *Manual de Usuario*, y se también ofrece ayuda desde el propio entorno de desarrollo mostrando la documentación relacionada con cada una

de las acciones que el usuario desee ejecutar. Además para facilitar aún más el proceso de creación de los scripts se ofrecen dos plantillas, una para definir nuevas *Baterías de Test* y otra para los *Casos de Test.*

Pero la principal característica de *X-Aute* es la de poder ejecutar dichos scripts de test y accionar automáticamente los controles de la *interfaz gráfica* que se desea testar. *X-Aute* es capaz de manejar los controles más comunes de las interfaces de usuario como menús, botones, listboxes etc. así como también los controles personalizados.

Todo esto convierte a *X-Aute* en una herramienta de gran utilidad pues la ejecución de las pruebas podrá realizarse de una manera automatizada, en el momento que convenga, por ejemplo y si se desea durante la noche, cuando las máquinas estén ociosas. Permite de una manera sencilla realizar pruebas de regresión tras haber modificado alguna parte del código debido a la corrección de un "*bug*", a la incorporación de una nueva funcionalidad, etc. o para realizar una revisión completa del correcto funcionamiento de todas las funcionalidades de la interfaz.

No obstante el solo hecho de accionar lo controles automáticamente no sirve de nada si el resultado de dichas acciones no queda reflejado en ningún sitio. *X-Aute* almacena en un fichero log cada una de las operaciones con los controles y el resultado de su ejecución, y poder de esto modo sacar las conclusiones acerca del correcto funcionamiento o no de la interfaz.

8. HERRAMIENTAS

Este capítulo contiene la descripción de las herramientas en las que se apoya el desarrollo del presente proyecto.

La fase de construcción de la herramienta *X-Aute* se ha sustentado en el conjunto de herramientas que se detallan a continuación y que son las siguientes: *Microsoft Visual Studio 2005®*, y el *.NET Framework®*.

8.1 MICROSOFT VISUAL STUDIO 2005®

Visual Studio 2005 es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones Web ASP.NET, servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Los lenguajes de programación Visual Basic .NET, Visual C++, Visual C# y Visual J# utilizan el mismo entorno de desarrollo integrado (IDE), que les permite compartir herramientas y facilita la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes aprovechan las funciones de .NET Framework, que ofrece acceso a tecnologías clave para simplificar el desarrollo de aplicaciones Web ASP y servicios Web XML.

8.2 .NET FRAMEWORK

.NET Framework es un entorno multilenguaje que permite generar, implantar y ejecutar aplicaciones y servicios Web XML. Es considerado como un componente de Windows para la creación y la ejecución de aplicaciones construidas en un amplio abanico de lenguajes de programación. Además, se encarga de proporcionar la mayor parte de la estructura necesaria para la generación del software, lo que permite que los desarrolladores se centren en el código lógico específico de su producto. En el marco de este proyecto, las tecnologías

proporcionadas por *.NET Framework* se aplican en la construcción del sistema software que soporta la plataforma *X-Aute*.

.NET Framework está compuesto por el CLR (Common Language Runtime) y por un conjunto unificado de biblioteca de clases. El CLR desempeña una función tanto durante la ejecución como durante el desarrollo de los componentes. Cuando el componente se está ejecutando, el motor en tiempo de ejecución es responsable de administrar la asignación de memoria, iniciar y detener procesos y subprocesos, y hacer cumplir la directiva de seguridad, así como satisfacer las posibles dependencias del componente sobre otros componentes.

Las bibliotecas de clases (clases de programación unificadas) proporcionan funciones estándar, como las de entrada/salida, manipulación de cadenas, administración de seguridad, comunicaciones y funciones de diseño del interfaz de usuario. Un subconjunto de clases de gran relevancia dentro de la biblioteca de clases de .NET Framework son las clases ADO .NET. Estas clases permiten a los programadores interactuar con los datos obtenidos a través de los interfaces de acceso a bases de datos ODBC, OLE DB, SQL Server, etc.

El framework de accesibilidad *UI Automation,* contenido dentro del *.Net Framework*, es el que permite a *X-Aute* localizar los controles e invocar las acciones, que se pretenden automatizar, sobre ellos, pero éste queda definido ampliamente dentro del capítulo *Desarrollo e Implementación de X-Aute.* (*Ver apartado 10.2.1 Framework UI Automation*).

9. METODOLOGÍA DE DESARROLLO

Para el desarrollo de este proyecto se ha aplicado una adaptación de los *Estándares de Ingeniería de Software ESA (European Space Agency, Agencia Espacial Europea),* cuya referencia es *PSS-05-0*. Estos estándares están especialmente indicados para proyectos de gran envergadura, por ello en vez de utilizar esta versión completa, se he empleado una versión denominada *PSS-05 lite* que está especialmente enfocada hacia proyectos software de pequeño tamaño. Según los estándares, un proyecto software pequeño es aquel que posee una o más de las siguientes características (ESA Board for Software Standardisation and Control, 1996):

- *Se necesitan menos de dos años hombre para su desarrollo.*
- *Se requiere un equipo único de desarrollo de cinco o menos personas.*
- *Tiene menos de 10.000 líneas de código, excluyendo los comentarios.*

La adaptación de los estándares a proyectos de menor tamaño se ha conseguido por medio de diversas estrategias entre las que se pueden destacar las siguientes (ESA Board for Software Standardisation and Control, 1996):

- *Combinación de los requisitos software y las fases de diseño arquitectónico.*
- *Elaboración de una documentación simplificada.*
- *Simplificación de los planes.*
- *Reducción de la formalidad de los requisitos.*
- *Utilización de las especificaciones del plan de pruebas para las pruebas de aceptación.*

En el resto de secciones de este capítulo se cubrirán los detalles más relevantes de la metodología aplicada, empezando por el modelo de ciclo de vida del proyecto y continuando después con una descripción de las diferentes fases del mismo.

9.1 CICLO DE VIDA

Uno de los principales aspectos de toda metodología de desarrollo software es su propuesta de ciclo de vida del proyecto. La guía de *PSS-05 lite* establece un ciclo de vida que divide el proyecto en cinco fases (ESA Board for Software Standardisation and Control, 1991):

- ***Fase RU:*** *Fase de definición de los requisitos de usuario.*
- ***Fase RS/DA:*** *Fases de definición de los requisitos software y del diseño arquitectónico.*
- ***Fase DD:*** *Fase de diseño detallado y de producción de código.*
- ***Fase TR:*** *Fase de transferencia de software.*
- ***Fase OM:*** *Fase de operación y mantenimiento.*

Para este proyecto se ha eliminado la última de las fases, la *fase OM*. Esta modificación se basa en el hecho de que el mantenimiento correctivo y evolutivo de la herramienta queda fuera del ámbito del presente proyecto fin de carrera. La ***figura 9.1*** es la representación gráfica del ciclo de vida de este proyecto, en el que las cajas azules son las fases del ciclo y en las blancas se expresan las salidas de cada una de ellas.

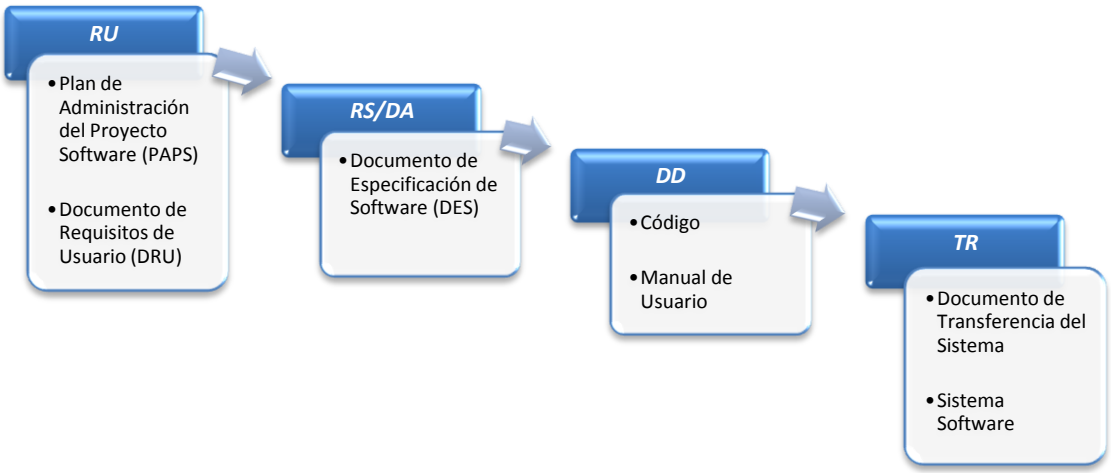


Figura 9.1. *Ciclo de vida del proyecto.*

9.2 FASE RU

La *fase RU* también se denomina *fase de definición del problema* y su propósito es convertir la idea inicial del proyecto en una definición precisa de las funcionalidades y las características que ha de tener el sistema una vez construido.

Esta fase necesita de una alta implicación del usuario para definir los requisitos del proyecto que también han de ser revisados y especificados de forma concreta por los ingenieros a cargo del proyecto. Para la extracción de dichos requisitos no se suele utilizar ningún método formal, sino que se extraen de entrevistas con los usuarios, encuestas, estudios, etc.

El resultado más importante de la *fase de RU* es el *Documento de Requisitos de Usuario o DRU*. Este documento es crítico para todo el posterior desarrollo del proyecto ya que sienta las bases para poder determinar si el sistema construido se adecúa a las peticiones del cliente.

La actividad principal de la fase *RU* es la captura de requisitos de usuario, los cuales serán documentados en el *DRU*. Además en dicha fase se llevarán a cabo las siguientes actividades (ESA Board for Software Standardisation and Control, 1991):

- ***Determinar el entorno operacional:*** En esta actividad se debe estudiar el entorno en el que se implantará el nuevo sistema, teniendo en cuenta las interacciones con el resto de sistemas que lo conforman.
- ***Especificar los requisitos de usuario****:* Después de establecer el contexto del sistema, se extraen los requisitos de usuario, intentando evitar al máximo cualquier detalle de implementación al menos que el requisito tenga específicamente dicha naturaleza.
- ***Clasificar los requisitos de usuario:*** Los requisitos de usuario se pueden dividir en dos categorías: capacidades y restricciones. Las capacidades son las necesidades del usuario para resolver un problema o alcanzar un objetivo, mientras que las restricciones son limitaciones en la manera en que se ha de resolver el problema.
- ***Asignar los atributos a los requisitos de usuario****:* Cada uno de los requisitos de usuario del sistema ha de estar etiquetado con los siguientes atributos:

    - *Identificador.* Todo requisito ha de tener un identificador único para permitir su trazabilidad e identificación en fases posteriores de desarrollo.
    - *Necesidad.* Atributo que mide el grado de importancia de que el requisito de usuario esté incorporado al sistema.
    - *Prioridad.* La prioridad de los requisitos de usuario sirve para establecer el orden de implementación de los mismos en los casos en los que haya un desarrollo evolutivo.
    - *Estabilidad.* La estabilidad de un requisito expresa el grado en el cual el requisito puede variar a lo largo del tiempo, por ejemplo, porque esté asociado a una determinada normativa o estándar.
    - *Fuente.* En equipos de trabajo formados por varias personas sirve para determinar de manera concreta quién introdujo el requisito asociado en la especificación.

- *Claridad.* Indica la falta de ambigüedad del requisito, entendida como la variedad de interpretaciones que se pueden dar del mismo.
- *Verificabilidad.* Este atributo expresa si es posible verificar que el requisito esté incorporado en el sistema construido.
- *Descripción.* Todo requisito debe ir acompañado de una descripción textual del mismo tan detallada como sea posible.

- ***Revisar los resultados de la fase.*** La revisión y aprobación por parte de todos los interesados del *DRU* simbolizan la finalización de la fase *RU* y el comienzo de la fase *RS/DA*, de la cual, el *DRU* es su principal entrada de información.

El futuro de todas estas tareas es la creación del *DRU*, que proporciona una visión general de lo que el usuario espera del sistema que se va a construir. El *DRU* también debe describir las operaciones que el usuario quiere realizar con el sistema así como las restricciones que, por parte del usuario, tengan que ser impuestas a la solución. El control de cambios del *DRU* ha de ser controlado por el usuario y toda modificación en el número o en el contenido de los requisitos ha de ser comunicada al equipo de trabajo y recogida en el propio documento (ESA Board for Software Standardisation and Control, 1991).

9.3 FASE RS/DA

La fase *RS/DA* es la combinación de las fases *RS* y *DA* del estándar *PSS-05-0*. La fase *RS* también se denomina *fase de análisis del problema* y su principal propósito es analizar los requisitos de usuario presentes en el *DRU* y transformarlos en un conjunto de requisitos software de la manera más completa, consistente y correcta posible. En esta fase deberían de participar tanto el usuario como los ingenieros del proyecto y el personal de operaciones ya que cada uno de estos grupos tienen un punto de vista diferente del sistema que ha de ser incorporado al desarrollo evitando inconsistencias entre los demás.

Por su parte, la fase *DA* es la *fase de la solución* dentro del ciclo de vida y su objetivo principal es la definición de un conjunto de componentes software y de los interfaces que permiten su intercomunicación para la construcción de un *framework* que permita el desarrollo del software del sistema.

La combinación de las dos fases en la fase *RS/DA* propuesta por *PSS-05 lite* se lleva a cabo siguiendo las estrategias planteadas al principio de este capítulo. Una consecuencia inmediata de dichas estrategias es la desaparición del *DDA* (*Documento de Diseño de Arquitectura*) y del *DDD*

(*Documento de Diseño Detallado*). En su lugar se introduce un nuevo documento, cuyas siglas son *DES* (*Documento de Especificación de Software*), que contiene los principales resultados de la fase *RS/DA* (ESA Board for Software Standardisation and Control, 1991).

El conjunto de actividades que han de ser desarrolladas en la fase *RS/DA* también son una combinación de las tareas correspondientes a las fases *RS* y *DA* por separado, y en concreto son (ESA Board for Software Standardisation and Control, 1991):

- ***Construcción del modelo lógico***. Se debe construir un modelo que recoja las necesidades del usuario que sea lo más independiente de la implementación posible. Este modelo se debe emplear principalmente para la extracción de los requisitos software. Un buen modelo lógico debería estar formado por funciones con propósitos diferenciados, con un nivel de abstracción adecuado a aquél en el que aparecen y bien definidas en cuanto a sus atributos de rendimiento. Debería minimizar los interfaces para evitar el acoplamiento entre componentes y evitar información relativa a la implementación.

- ***Especificación de los requisitos software***. Es labor del equipo de ingenieros la extracción de los requisitos software a partir del modelo lógico. El conjunto de requisitos enumerados ha de ser completo y consistente. Entendiendo la completitud como el hecho de que tienen en cuenta todos y cada uno de los requisitos de usuario y cubren cualquier conjunto de posibles entradas al sistema; y la consistencia como el hecho de que ningún par de requisitos estén en conflicto. Los requisitos software han de ser clasificados dentro de las siguientes categorías: *funcionales, de rendimiento, de interfaz, operativos, de recursos, de verificación, de test de aceptación, de documentación, de seguridad, de portabilidad, de calidad, de fiabilidad y de mantenimiento*. Al igual que pasaba con los requisitos de usuario, los requisitos software han de ser etiquetados con un conjunto de atributos que los describan. En este caso, los atributos necesarios coinciden con los utilizados en la especificación de los requisitos de usuario.

- ***Construcción del modelo físico***. El modelo físico es una descripción del diseño del sistema empleando terminología asociada a la implementación. La transformación del modelo lógico al modelo físico se hace distribuyendo las funciones identificadas entre los distintos compnentes y definiendo las entradas y las salidas de dichas funciones. Es muy recomendable que la descomposición en componentes sea jerárquica, para permitir la encapsulación de la información y conseguir, en el nivel más bajo, un conjunto de componentes suficientemente independientes que puedan ser especificados en detalle y codificados de manera paralela.

- ***Especificación del diseño arquitectónico***. El diseño arquitectónico es el modelo físico documentado de manera completa, preferiblemente por medio de diagramas que representen la interacción entre los componentes de los distintos niveles de la arquitectura. El resultado de este proceso es un conjunto de componentes que tienen perfectamente definidos tanto sus datos de entrada y salida como las funciones que proporcionan.

La finalización de estas actividades da lugar al *Documento de Especificación de Software* que será utilizado como entrada en la siguiente fase del ciclo de vida del proyecto, la fase *DD* o de *Diseño Detallado*.

9.4 FASE DD

La fase *DD* es la *fase de implementación* dentro del ciclo de vida. El objetivo de esta fase es detallar el diseño planteado durante la fase *RS/DA* para codificarlo, documentarlo y probarlo (ESA Board for Software Standardisation and Control, 1991).

Las actividades correspondientes a esta fase se pueden reducir esencialmente a dos, la de elaboración del *diseño detallado* y la de *producción*, que se describen a continuación:

- ***Diseño detallado.*** El diseño detallado consiste en la descomposición de los componentes del diseño arquitectónico hasta el punto en el que puedan ser expresados como módulos, o unidades independientes de programación, del lenguaje de programación en el que vaya a ser construido el sistema.

- ***Producción.*** La actividad de producción consiste en la codificación, la integración y la prueba de los módulos generados en el diseño detallado.

La fase *DD* da lugar a dos productos de salida: el *código del sistema* y *MUS* o *Manual de Usuario*. El *código fuente* ha de contener el *diseño detallado* de cada uno de los módulos que lo compongan aunque este diseño también se puede incluir dentro del *DES*. El *manual de usuario* consiste en una guía acerca de la utilización del sistema. Estará dividido en dos partes con estilos bien diferenciados. La primera parte está orientada a los nuevos usuarios del sistema y a modo de tutorial muestra en primer lugar las operaciones más sencillas que el usuario puede realizar, para después ir abordando las operaciones más avanzadas y complejas. La segunda parte es una referencia formal de todas las operaciones del sistema.

9.5 FASE TR

La fase *TR* también se denomina *fase de entrega*. El propósito de esta fase es la instalación, en el entorno operativo, del sistema construido y demostrar a los interesados que cumple con las características recogidas en el *DRU*:

Esta fase tiene dos actividades principales que son las que se describen a continuación:

- ***Instalación***. Para la instalación del sistema, primero hay que comprobar que los entregables proporcionados se corresponden a la lista de elementos de configuración necesarios para la instalación, para después implantar el nuevo sistema en su entorno operativo.

- ***Pruebas de aceptación***. Las pruebas de aceptación sirven para demostrar las capacidades del sistema implantado. Las pruebas de aceptación han de basarse en los requisitos de usuario recogidos en el *URD*. Los resultados de las pruebas se incluirán en el *DTS* (*Documento de Transferencia de Software*).

La conclusión de la fase *TR* da lugar a dos productos: el *sistema construido* y el *DTS*. Los contenidos del *DTS* son tales que permitan identificar el software transferido y enuncien de manera detallada cómo instalarlo.

9.6 RESUMEN

El desarrollo de este proyecto sigue un ciclo de vida basado en el estándar *PSS-05 lite* de la *Agencia Espacial Europea*. De la ejecución de cada una de las fases de la metodología propuesta se obtienen un conjunto de entregables que se detallan a continuación:

- ***Fase RU***. *Documento de Requisitos de Usuario (DRU).*
- ***Fase RS/DA***. *Documento de Especificación de Software* (*DES*).
- ***Fase DD***. *Código fuente del sistema* y *Manual de Usuario* (*MUS*).
- ***Fase TR***. *El sistema software construido y el Documento de Transferencia de Sistema (DTS).*

El Documento de Requisitos de Usuario, *DRU*, se incluye como apéndice a este documento del Proyecto Fin de Carrera.

10. PAQUETES DE TRABAJO, CALENDARIO Y PRESUPUESTO

Esta sección presenta la descomposición de las tareas del proyecto en paquetes de trabajo, el calendario que marca la evolución de las mismas y el presupuesto para la ejecución del proyecto.

10.1 PAQUETES DE TRABAJO

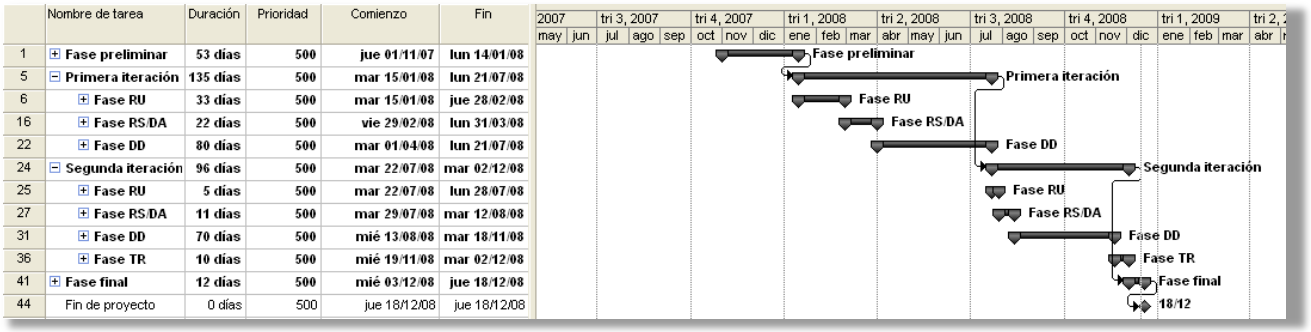
De la descomposición de las fases que componen el ciclo de vida del proyecto se han obtenido los paquetes de trabajo que se muestran en la siguiente tabla.

FASE	NOMBRE DEL PAQUETE
Fase Preliminar	*Introducción* *Objetivos* *Estado del Arte.*
Fase RU	*Documento de Requisitos de Usuario* *Plan de Administración del Proyecto Software*
Fase RS/DA	*Documento de Especificación Software*
Fase DD	*Codificación* *Manual de Usuario*
Fase TR	*Documento de Transferencia Software*
Fase Final	*Revisión General* *Preparación de la defensa*

Tabla 10.1. *Paquetes de trabajo.*

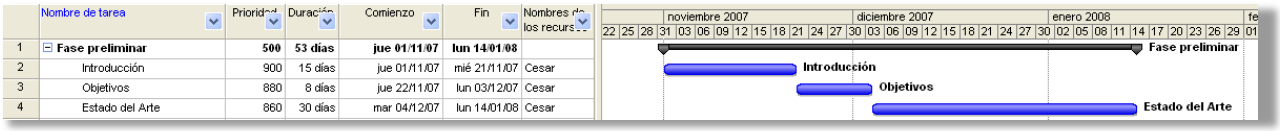
10.2 CALENDARIO

Las figuras 10.1 y 10.2 muestras el calendario del proyecto descompuesto en sus tareas principales y en todas sus tareas, respectivamente. Las figuras 10.2, 10.3, 10.4 y 10.5 muestran de manera más detallada la planificación de cada una de estas tareas.



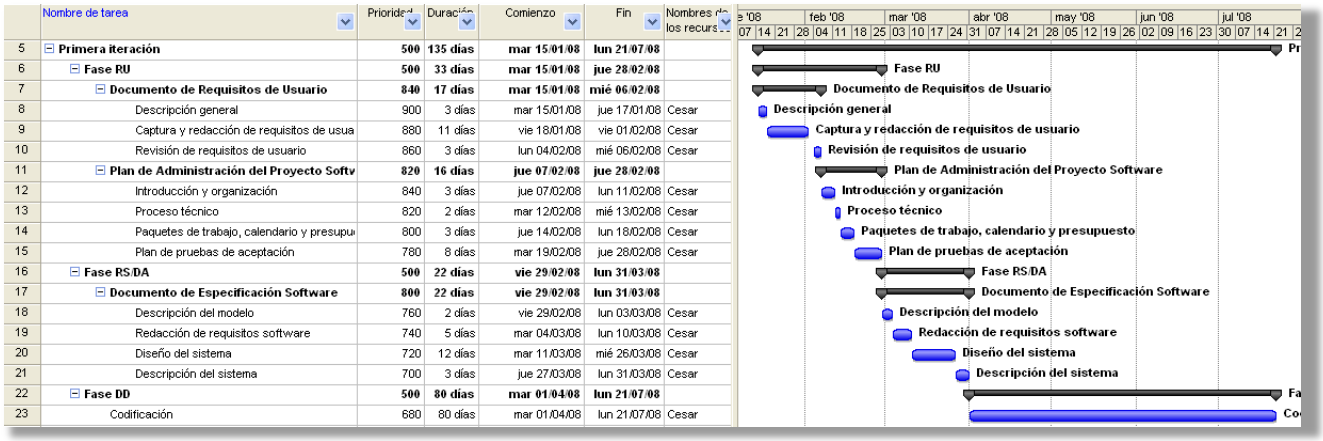
	Nombre de tarea	Duración	Prioridad	Comienzo	Fin
1	Fase preliminar	53 días	500	jue 01/11/07	lun 14/01/08
5	Primera iteración	135 días	500	mar 15/01/08	lun 21/07/08
6	Fase RU	33 días	500	mar 15/01/08	jue 28/02/08
16	Fase RS/DA	22 días	500	vie 29/02/08	lun 31/03/08
22	Fase DD	80 días	500	mar 01/04/08	lun 21/07/08
24	Segunda iteración	96 días	500	mar 22/07/08	mar 02/12/08
25	Fase RU	5 días	500	mar 22/07/08	lun 28/07/08
27	Fase RS/DA	11 días	500	mar 29/07/08	mar 12/08/08
31	Fase DD	70 días	500	mié 13/08/08	mar 18/11/08
36	Fase TR	10 días	500	mié 19/11/08	mar 02/12/08
41	Fase final	12 días	500	mié 03/12/08	jue 18/12/08
44	Fin de proyecto	0 días	500	jue 18/12/08	jue 18/12/08

Figura 10.1. *Calendario del proyecto.*



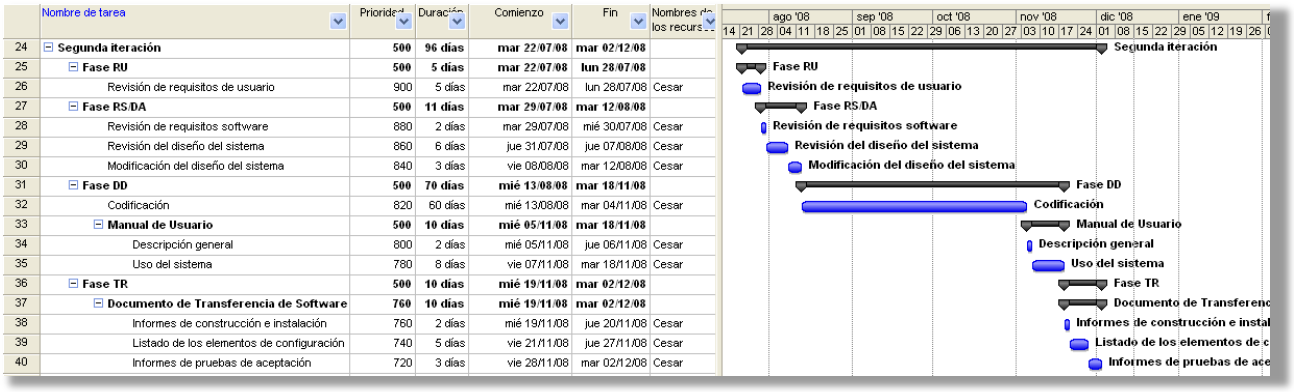
	Nombre de tarea	Prioridad	Duración	Comienzo	Fin	Nombres de los recurs
1	Fase preliminar	500	53 días	jue 01/11/07	lun 14/01/08	
2	Introducción	900	15 días	jue 01/11/07	mié 21/11/07	Cesar
3	Objetivos	880	8 días	jue 22/11/07	lun 03/12/07	Cesar
4	Estado del Arte	860	30 días	mar 04/12/07	lun 14/01/08	Cesar

Figura 10.2. *Fase preliminar.*



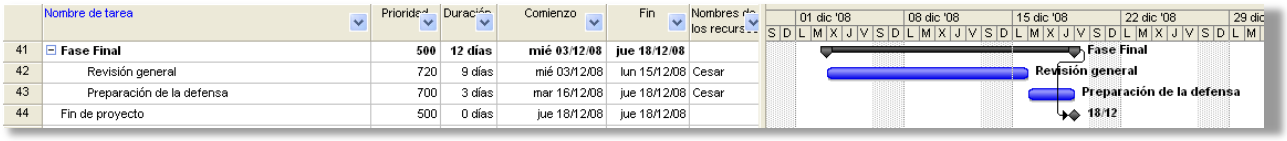
	Nombre de tarea	Prioridad	Duración	Comienzo	Fin	Nombres de los recurs
5	Primera iteración	500	135 días	mar 15/01/08	lun 21/07/08	
6	Fase RU	500	33 días	mar 15/01/08	jue 28/02/08	
7	Documento de Requisitos de Usuario	840	17 días	mar 15/01/08	mié 06/02/08	
8	Descripción general	900	3 días	mar 15/01/08	jue 17/01/08	Cesar
9	Captura y redacción de requisitos de usua	880	11 días	vie 18/01/08	vie 01/02/08	Cesar
10	Revisión de requisitos de usuario	860	3 días	lun 04/02/08	mié 06/02/08	Cesar
11	Plan de Administración del Proyecto Softv	820	16 días	jue 07/02/08	jue 28/02/08	
12	Introducción y organización	840	3 días	jue 07/02/08	lun 11/02/08	Cesar
13	Proceso técnico	820	2 días	mar 12/02/08	mié 13/02/08	Cesar
14	Paquetes de trabajo, calendario y presupu	800	3 días	jue 14/02/08	lun 18/02/08	Cesar
15	Plan de pruebas de aceptación	780	8 días	mar 19/02/08	jue 28/02/08	Cesar
16	Fase RS/DA	500	22 días	vie 29/02/08	lun 31/03/08	
17	Documento de Especificación Software	800	22 días	vie 29/02/08	lun 31/03/08	
18	Descripción del modelo	760	2 días	vie 29/02/08	lun 03/03/08	Cesar
19	Redacción de requisitos software	740	5 días	mar 04/03/08	lun 10/03/08	Cesar
20	Diseño del sistema	720	12 días	mar 11/03/08	mié 26/03/08	Cesar
21	Descripción del sistema	700	3 días	jue 27/03/08	lun 31/03/08	Cesar
22	Fase DD	500	80 días	mar 01/04/08	lun 21/07/08	
23	Codificación	680	80 días	mar 01/04/08	lun 21/07/08	Cesar

Figura 10.3. *Primera Iteración.*



	Nombre de tarea	Prioridad	Duración	Comienzo	Fin	Nombres de los recurs
24	**Segunda iteración**	**500**	**96 días**	**mar 22/07/08**	**mar 02/12/08**	
25	**Fase RU**	**500**	**5 días**	**mar 22/07/08**	**lun 28/07/08**	
26	Revisión de requisitos de usuario	900	5 días	mar 22/07/08	lun 28/07/08	Cesar
27	**Fase RS/DA**	**500**	**11 días**	**mar 29/07/08**	**mar 12/08/08**	
28	Revisión de requisitos software	880	2 días	mar 29/07/08	mié 30/07/08	Cesar
29	Revisión del diseño del sistema	860	6 días	jue 31/07/08	jue 07/08/08	Cesar
30	Modificación del diseño del sistema	840	3 días	vie 08/08/08	mar 12/08/08	Cesar
31	**Fase DD**	**500**	**70 días**	**mié 13/08/08**	**mar 18/11/08**	
32	Codificación	820	60 días	mié 13/08/08	mar 04/11/08	Cesar
33	**Manual de Usuario**	**500**	**10 días**	**mié 05/11/08**	**mar 18/11/08**	
34	Descripción general	800	2 días	mié 05/11/08	jue 06/11/08	Cesar
35	Uso del sistema	780	8 días	vie 07/11/08	mar 18/11/08	Cesar
36	**Fase TR**	**500**	**10 días**	**mié 19/11/08**	**mar 02/12/08**	
37	**Documento de Transferencia de Software**	**760**	**10 días**	**mié 19/11/08**	**mar 02/12/08**	
38	Informes de construcción e instalación	760	2 días	mié 19/11/08	jue 20/11/08	Cesar
39	Listado de los elementos de configuración	740	5 días	vie 21/11/08	jue 27/11/08	Cesar
40	Informes de pruebas de aceptación	720	3 días	vie 28/11/08	mar 02/12/08	Cesar

Figura 10.4. *Segunda Iteración.*



	Nombre de tarea	Prioridad	Duración	Comienzo	Fin	Nombres de los recurs
41	**Fase Final**	**500**	**12 días**	**mié 03/12/08**	**jue 18/12/08**	
42	Revisión general	720	9 días	mié 03/12/08	lun 15/12/08	Cesar
43	Preparación de la defensa	700	3 días	mar 16/12/08	jue 18/12/08	Cesar
44	Fin de proyecto	500	0 días	jue 18/12/08	jue 18/12/08	

Figura 10.5. *Fase Final.*

10.3 PRESUPUESTO

Este apartado detalla el presupuesto, tanto de tiempo como monetario, del desarrollo del proyecto *X-Aute*.

En primer lugar se presenta el análisis del coste temporal invertido en la realización del proyecto y en el apartado 10.3.2 se especifica el coste monetario del mismo.

10.3.1 Análisis del coste temporal

Tal y como se muestra en el calendario mostrado en el anterior apartado, el desarrollo de este proyecto se extiende desde el 01 de Noviembre de 2007 hasta el momento de la defensa del mismo el 18 de Diciembre de 2008. La siguiente tabla presenta las horas invertidas en cada una de las fases del proyecto durante este periodo de tiempo.

FASE	HORAS
Fase Preliminar	260 h.
Fase RU	180 h.
Fase RS/DA	160 h.
Fase DD	280 h.
Fase TR	20 h.
Fase Final	50 h.
TOTAL	**950 h**.

Tabla 10.2. *Coste temporal.*

10.3.2 Análisis del coste monetario

Para determinar el coste monetario del proyecto se ha de tener en cuenta tanto el coste del software utilizado como el coste de los recursos humanos. En este caso, el coste del software es nulo puesto que todas las herramientas utilizadas han sido adquiridas a través de programas / convenios por los que la Universidad Carlos III pone a disposición de sus alumnos ciertas herramientas informáticas para su uso académico, o bien son herramientas sin coste monetario asociado.

De acuerdo con el análisis de coste temporal realizado en el apartado anterior, la siguiente tabla detalla el coste monetario por horas de los diferentes roles implicados en el desarrollo del proyecto:

ROL	DEDICACIÓN	COSTE POR HORA	TOTAL
Investigador	310 h.	20 €/h.	6200 €
Analista	200 h.	40 €/h.	8000 €
Diseñador	160 h.	40 €/h.	6400 €
Programador	280 h.	25 €/h.	7000 €
		TOTAL	***27600 €***

Tabla 10.3. *Coste monetario.*

11. DESARROLLO E IMPLEMENTACIÓN DE X-AUTE

En este capítulo se explican todos los detalles referentes al desarrollo e implementación de la herramienta *X-Aute*.

X-Aute ha sido implementado utilizando el lenguaje C# y aprovechando las características y ventajas que ofrece la programación orientada a objetos como son la herencia, el polimorfismo o la encapsulación de datos, entre otras.

11.1 DESCRIPCIÓN Y MODELO DEL SISTEMA

Para diseñar la herramienta *X-Aute* se ha modelado cada posible control de una interfaz *WPF,* abstrayendo su comportamiento y características esenciales, y se ha representado en un clase C#. Para comprender mejor esto y hacerse una idea del diseño e implementación de la herramienta lo primero que se debe hacer es mostrar un diagrama del sistema desarrollado. En la Figura 10.1 se muestra el diagrama de clases de *X-Aute.*

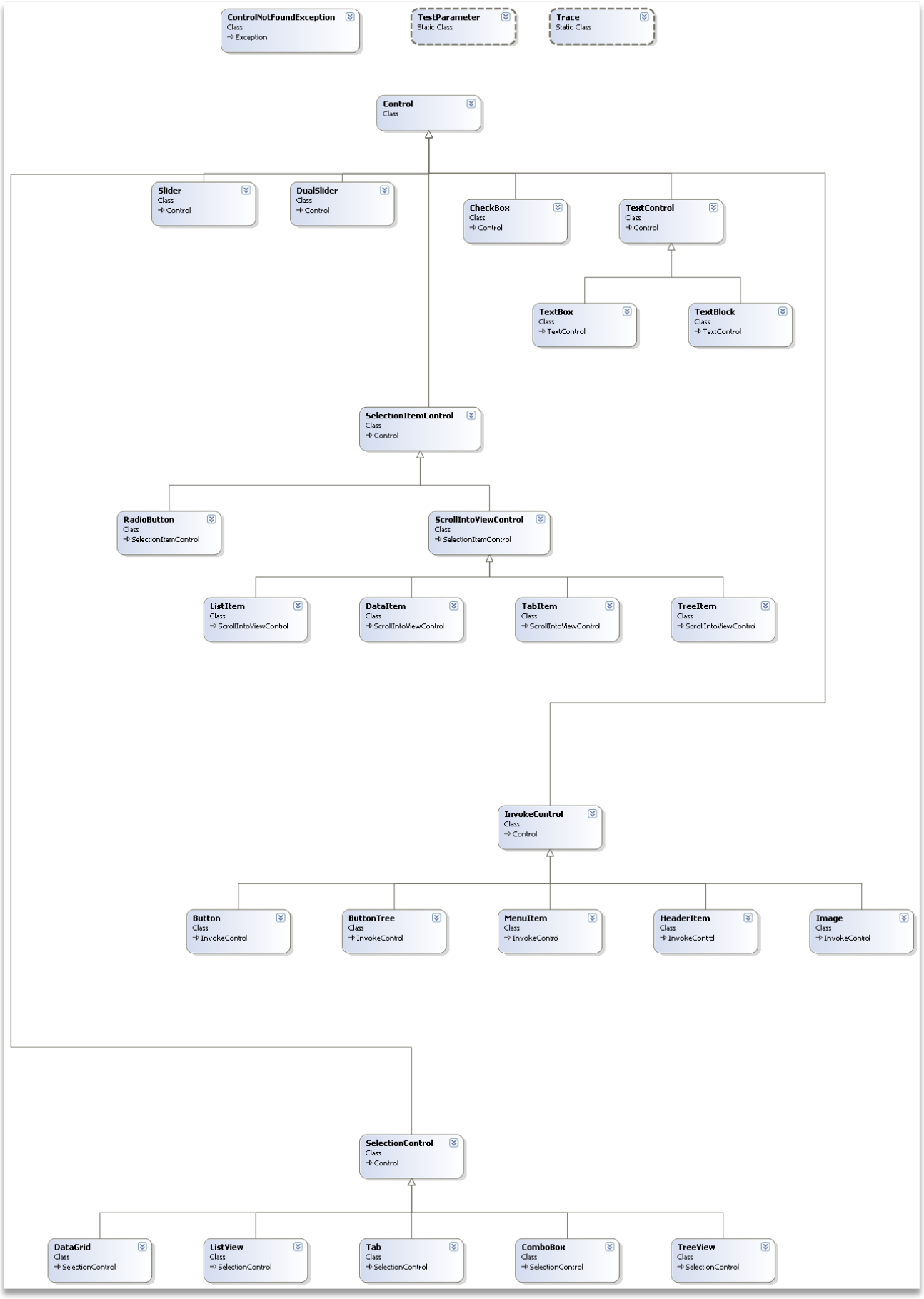


Figura 11.1. *Diagrama de clases de X-Aute.*

Es esta imagen queda muy claro el uso de una de las características principales de la programación orientada a objetos, la herencia. Las características y comportamientos comunes entre los controles están almacenados en niveles superiores, son clases padre, y a medida que se va bajando por el árbol de herencia se alcanzan las clases que modelan controles cada vez más detallados. De este modo se evita repetir el mismo código múltiples veces y se favorece la reutilización.

En los niveles intermedios del árbol de herencia están las clases que almacenan la funcionalidad compartida entre grupos de objetos y que es propia de los controles que modelan, situación que no sucedía con los comportamientos del nivel superior, es decir, que son acciones que el usuario podrá invocar con ese control. Por ejemplo el control botón tiene una funcionalidad propia que es el hecho de poder invocarlo pero no el de buscar una ventana o un botón más allá de que esto sea necesario para poder realizar lo primero.

Por último, todos los elementos hoja del árbol son las clases que pueden ser utilizadas por el usuario que va a escribir el test para realizar las acciones automatizadas sobre los controles que estas clases modelan. Cada instancia de una de estas clases es un objeto que representa un tipo de control de las interfaces *WPF* cuyo testeo se quiere automatizar.

11.1.1 Clase Control

La clase ***Control*** es padre de todas las clases implementadas, que representan controles, y en ella están definidas las características y comportamientos comunes que posen todos y cada uno de los controles modelados.

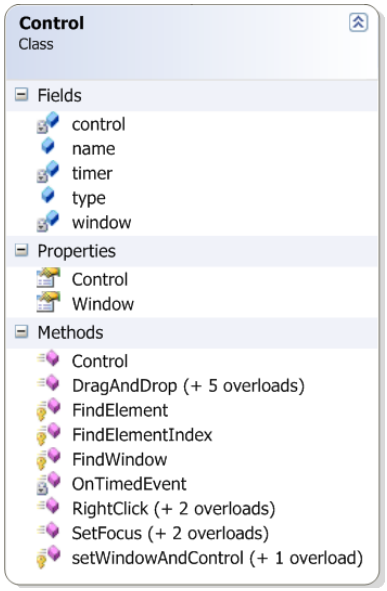


Figura 11.2. *Clase Control.*

Dos de las funcionalidades que ofrece ***Control*** y que han de poseer todos son hijos son la búsqueda de la ventana y la búsqueda del control. Esto es debido a que para poder realizar cualquier acción sobre un control primero es necesario localizar dicho objeto. Pero para poder encontrarlo es necesario inicialmente localizar cual es la ventana que lo contiene y después buscar sobre los objetos contenidos por ésta hasta dar con el control y ejecutar la acción que el usuario desee.

En caso que la búsqueda de la ventana y el control no se resuelva con éxito y no se encuentre alguno de los elementos demandados se lanza una excepción: ***ControlNotFoundException,*** que después la herramienta *X-Aute* se encargará de controlar y poder de esta manera continuar con el flujo de ejecución del test automatizado sin que se produzca una parada inesperada. Además dicha excepción, producto del resultado fallido en la búsqueda de un control sobre el que el usuario quiere ejecutar alguna acción, se dejará informada en el fichero de log.

Existen más comportamientos comunes entre todos los objetos que están almacenados en la clase padre ***Control,*** establecer el foco y desplegar el menú contextual asociado al control (el que aparece al hacer clic secundario).

Sin embargo la funcionalidad *DragAndDrop* implementada también en la clase ***Control*** es algo especial. Pues si bien no todos los objetos son arrastrables (como parece indicar el hecho que esta función este implementada en la clase padre), tampoco se puede hacer una generalización de cuales sí lo son y cuáles no puesto que es posible, por ejemplo, que en alguna ventana de la interfaz los controles *DataItem* sean arrastrables mientras que en otras no. Junto con este motivo y que según la división por funcionalidades actual habría que colocar el comportamiento *DragAndDrop* en varias clases intermedias, en algunas clases hojas, es decir repetir absurdamente el mismo código sin llegar a asegurar siempre que el objeto que implementa dicha funcionalidad presente continuamente tal comportamiento, se ha decidido almacenar esta funcionalidad en la clase padre, quedando a cargo del usuario desarrollador de scripts hacer uso de tal funcionalidad únicamente en los casos que los controles realmente presenten ese comportamiento.

El resto de controles se van a ir explicando agrupados por las funcionalidades comunes que posean.

11.1.2 Grupo de controles Slider

El *Slider* es un control típico en interfaces de usuario, formado por una barra y un puntero deslizante que modifica su valor moviendo el puntero a través de la barra.

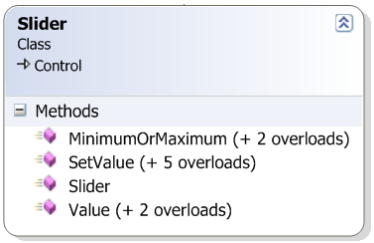


Figura 11.3. *Clase Slider.*

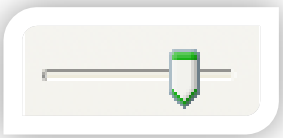


Figura 11.4. *Control Slider de una GUI.*

Las acción que el usuario podrá ejecutar automáticamente sobre estos controles es la de desplazar el slider estableciendo otro valor. Además existen otras funciones que permiten obtener el valor actual del slider y obtener sus valores máximo y mínimo.

El segundo de los controles el *DualSlider* podía hacer sido manejado perfectamente por el slider, puesto que este control no es más que la suma de dos slider, pero se ha incluido para mostrar como la herramienta es también capaz de interactuar con los controles personalizados, sin más que modelarlos en caso que se quiera hacer una invocación específica a ellos.

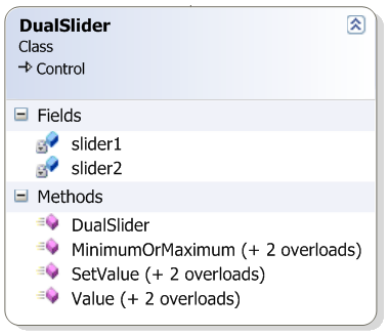


Figura 11.5. *Clase DualSlider.*

11.1.3 Grupo TextControl

Este grupo está formado por los controles relacionados con la captura o presentación de texto como son el control *TextBox* y *TextBlock*.

El *TextBox* es un cuadro que se utiliza para capturar el texto que introduzca el usuario mientras que el *TextBlock* se utiliza comúnmente para mostrar pequeños bloques de texto al usuario.

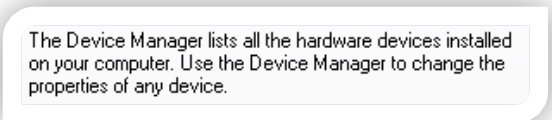


Figura 11.6. *Control TextBlock de una GUI.*

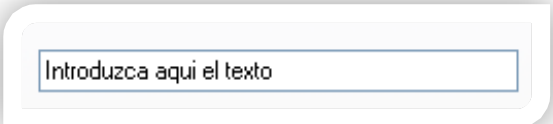


Figura 11.7. *Control TextBox de una GUI.*

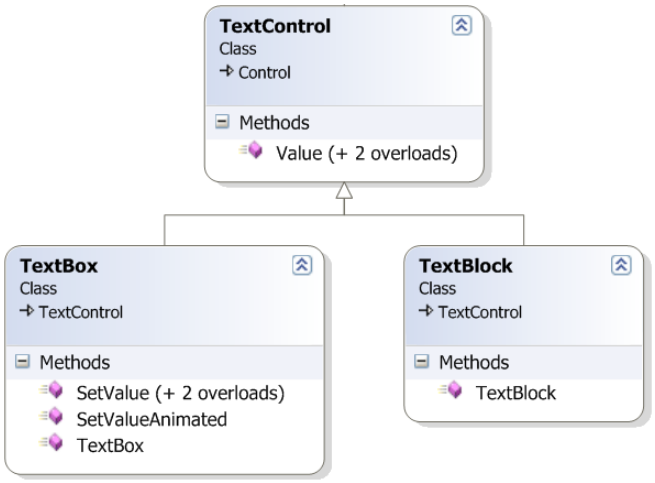


Figura 11.8. *Clases del grupo TextControl.*

TextControl es una clase intermedia, hija directa de la clase ***Control*** y padre de ***TextBox*** y ***TextBlock***. Almacena la funcionalidad común de sus clases hijas, que no es otra que la de obtener el texto que está mostrando el control en ese instante. Por otra parte el ***TextBox*** puede también insertar en la caja de texto los caracteres que el usuario le indique.

11.1.4 Control CheckBox

El *CheckBox* no comparte funcionalidad específica con ningún otro por lo que no forma parte de ningún grupo de controles.

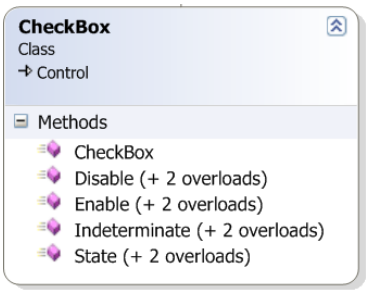


Figura 11.9. *Clase CheckBox.*

Figura 11.10. *Control CheckBox de una GUI.*

El comportamiento que almacena esta clase son las acciones típicas que se pueden realizar con este tipo de controles: marcar, desmarcar, establecer un estado intermedio, en caso de que se pueda, y consultar su estado actual.

11.1.5 Grupo SelectionItemControl

Este amplio grupo está formado por los controles que representan elementos de interfaz seleccionables, estos son: *RadioButton*, *DataItem*, *TreeItem*, *TabItem*, *ListItem* y por una clase intermedia más, ***ScrollIntoViewControl***

Figura 11.11. *Control RadioButton.*

Figura 11.12. *Control TreeItem.*

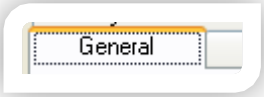


Figura 11.13. *Control TabItem.*

Como se puede apreciar en las imágenes (tan sólo se muestran algunos) estos controles presentan claras diferencias entre ellos pero a pesar de esto todos tienen una funcionalidad común, la poder ser seleccionados.

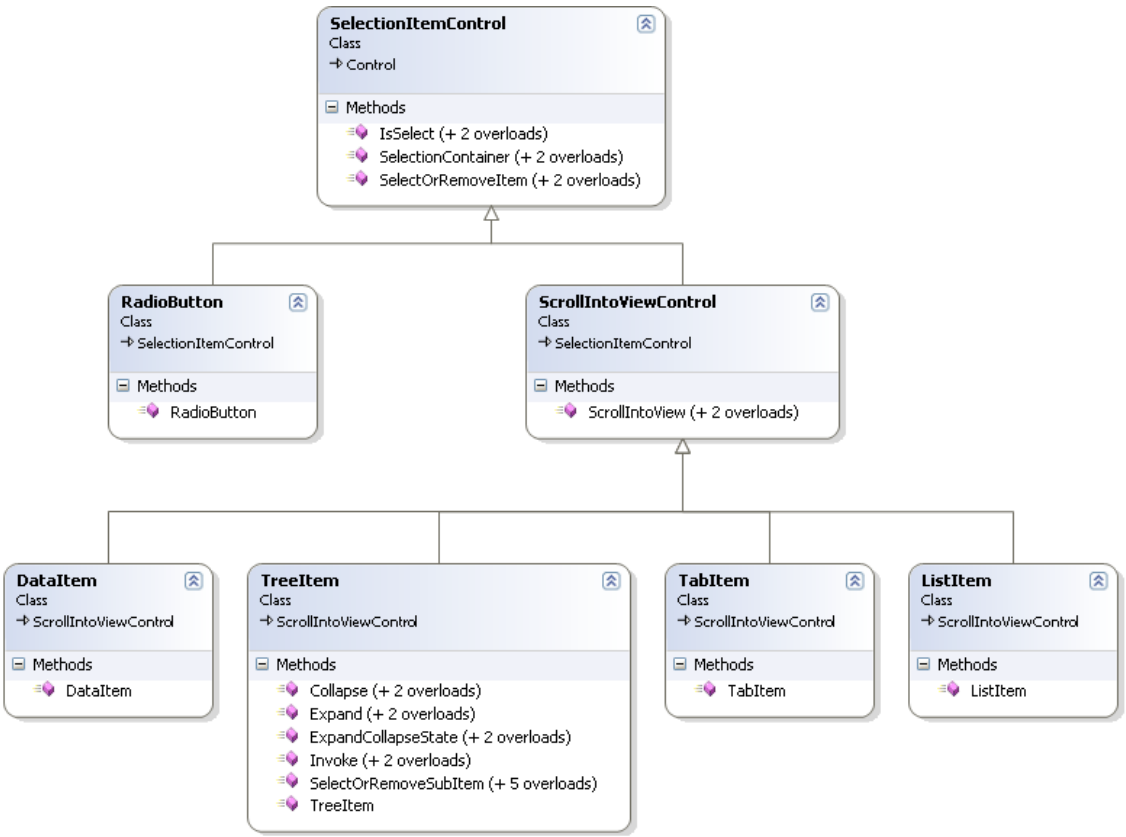


Figura 11.14. *Clases del grupo SelectionItemControl.*

El comportamiento común que almacena la clase ***SelectionItemControl***, hija directa de ***Control***, está relacionado con la selección de los elementos de interfaz. Concretamente las funcionalidades que almacena son la de consultar si un elemento concreto (un ítem) está seleccionado, la de obtener el contenedor que contiene dicho control - por ejemplo obtener la lista que contiene un determinado *ListItem* - y la de seleccionar un elemento o eliminarlo de la selección.

Además hay otra clase intermedia, ***ScrollIntoViewControl***, que almacena un comportamiento que es común para los controles que modelan las clases: ***DataItem***, ***TreeItem***, ***TabItem***, y ***ListItem***. Dicho comportamiento es la capacidad de realizar un scroll de la barra de desplazamiento asociada a su contenedor hasta conseguir que el ítem concreto sea visible en la pantalla.

Además de todas estas funcionalidades comunes los objetos *TreeItem* presenta las siguientes funcionalidades específicas: pueden ser expandidos o colapsados, se puede consultar el estado en que se encuentra (expandido o colapsado), puede buscar entre sus *TreeItem* hijos alguno que cumpla con los criterios de búsqueda que el usuario le indique y por último pueden ser invocados tratándoseles para ello como un botón.

11.1.6 Grupo SelectionControl

Este amplio grupo está compuesto por los controles que representan los contenedores de los objetos anteriores, DataGrid, *TreeView, Tab, ListView,* además del *ComboBox*.

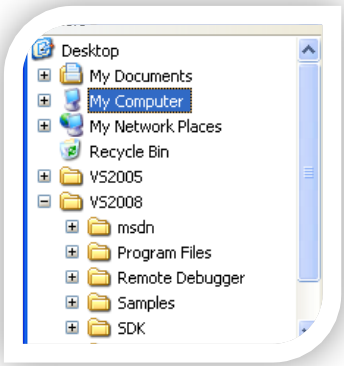


Figura 11.15. *Control TreeView.*

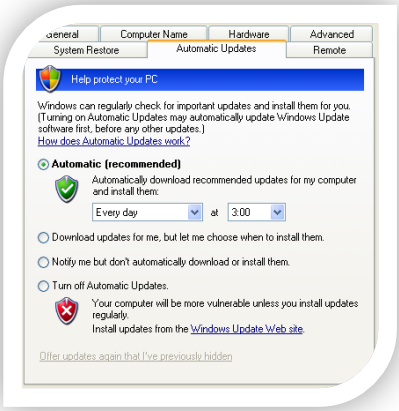


Figura 11.16. *Control Tab.*

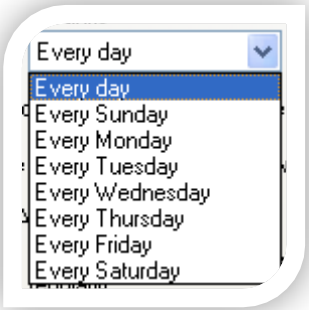


Figura 11.17. *Control ComboBox.*

Al igual que sucedía con el grupo anterior estos controles son muy diferentes entre sí pero tienen un comportamiento común que les une, todos son controles que pueden seleccionar a sus ítems hijos.

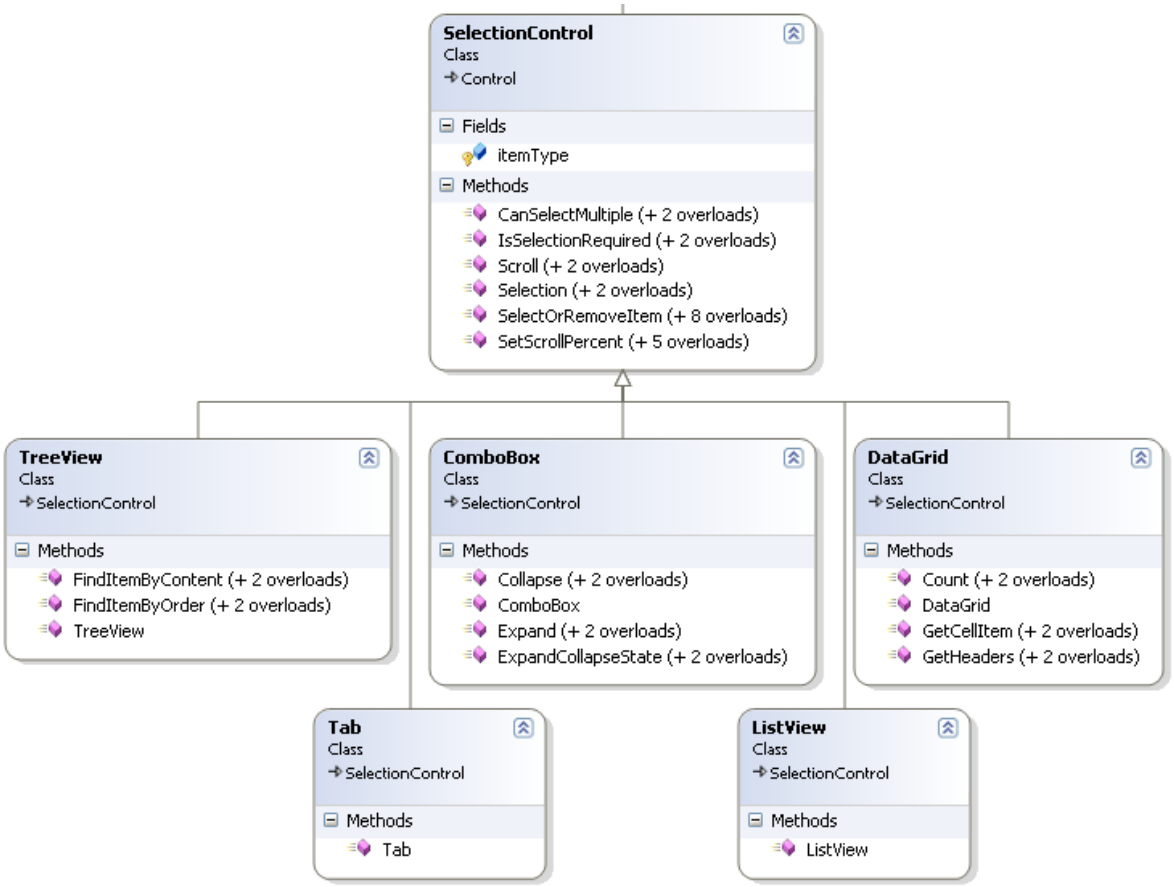


Figura 11.18. *Clases del grupo SelectionControl.*

El comportamiento común que almacena la clase ***SelectionControl,*** hija directa de ***Control,*** está relacionado con la selección de algunos de los elementos contenidos por el control que modelan las clases que heredan de esta. Concretamente las funcionalidades comunes son: consultar si el control soporta la selección múltiple, consultar si el control requiere que al menos un hijo este seleccionado, mover la barra de desplazamiento de diferentes maneras (especificando el porcentaje a desplazar o la cantidad), obtener una colección con todos los hijos seleccionados y el más importante seleccionar elementos hijos o eliminarlos de la selección.

Dentro de este grupo hay tres controles que presentan características específicas y que a continuación se enumeran:

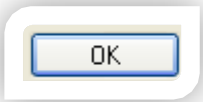
La clase ***TreeView*** permite buscar, entre sus elementos hijos, un *TreeItem* que contenga el texto indicado o que su ordinal sea el deseado.

La clase ***ComboBox*** permite expandir o colapsar el control y consultar el estado en que se encuentra (expandido o colapsado).

Por último, la clase ***DataGrid*** permite consultar el número total de filas y columnas que el control tiene, obtener el ítem de una celda en concreto (representada por número de fila y número de columna) y obtener una colección con todas las cabeceras del *DataGrid*.

11.1.7 Grupo InvokeControl

El último grupo de clases son las que modelan los controles que pueden ser invocados, *Button, ButtonTree, Image, MenuItem, HeaderItem*.



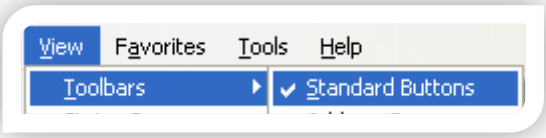


Figura 11.19. *Control Button.* ***Figura 11.20.*** *Control TreeButton.* ***Figura 11.21.*** *Control MenúItem.*

Al igual que sucedía en los dos últimos grupos, estos controles son muy diferentes entre sí pero tienen un comportamiento común que les une, todos son controles que se pueden invocar para provocar alguna acción en la interfaz.

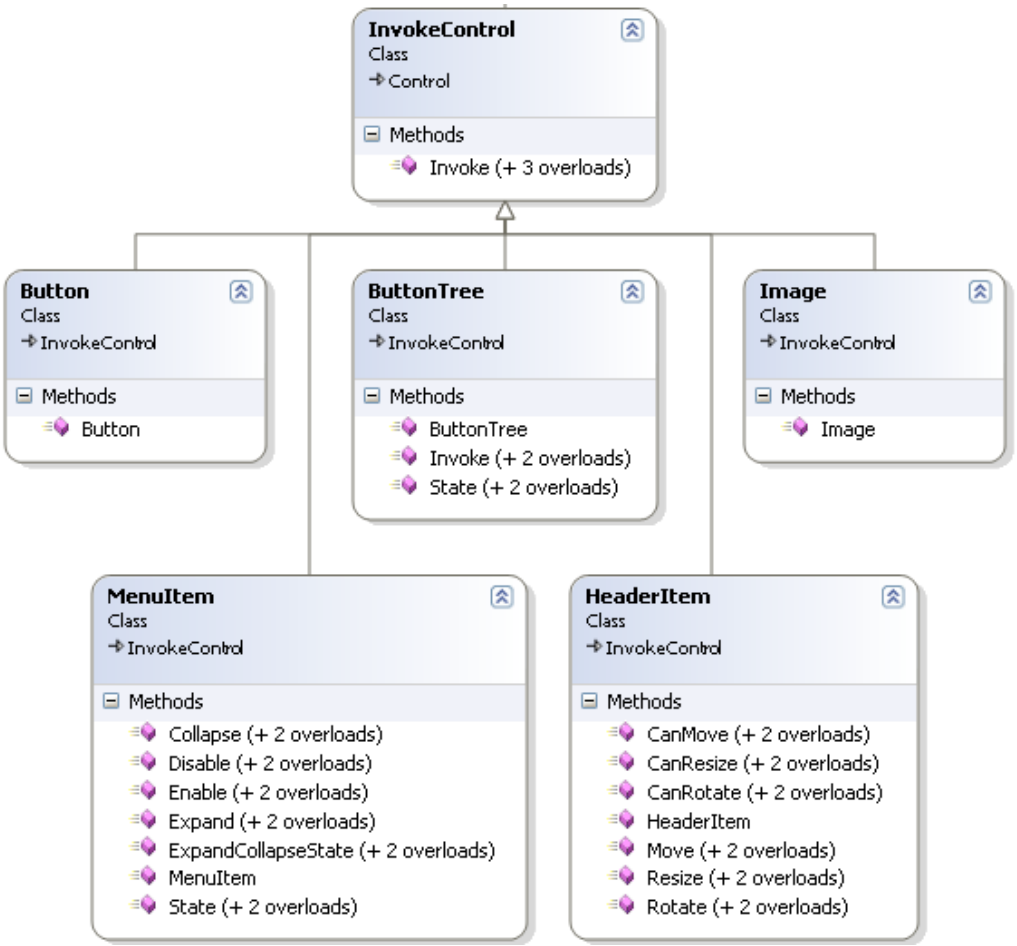


Figura 11.22. *Clases del grupo InvokeControl.*

El comportamiento común que almacena la clase ***InvokeControl,*** hija directa de ***Control,*** está relacionado con la posibilidad de invocar los controles para provocar alguna acción en la interfaz.

Dentro de este grupo hay tres controles que presentan características específicas y que a continuación se enumeran:

La clase ***ButtonTree,*** representa el *botón* de un *TreeItem* que permite ser manejado automáticamente. Permite expandir y colapsar el nodo de un árbol, además de poder consultar cual es su estado actual.

La clase ***HeaderItem*** representa un elemento de encabezado, del que se puede consultar si puede ser movido, rotado o redimensionado, además y en caso de que sea posible tales controles se pueden mover, rotar o redimensionar.

Por último la clase ***MenuItem*** representa un elemento de menú. Algunos de estos elementos de menú pueden ser expandidos, colapsados y consultados por su estado actual además de poderse marcar, desmarcar, establecer como marcado intermedio y consultar el estado actual, en esta ocasión si está marcado, desmarcado o intermedio.

11.2 Detalles de Implementación

Después de describir la clase ***Control*** y todas las demás agrupadas por funcionalidad se pueden explicar ciertos detalles de implementación que permitirán conocer y comprender mejor el modo de funcionamiento de la herramienta *X-Aute.*

En muchas ocasiones, los métodos que modelan el comportamiento de los controles presentan varias sobrecargas. Esto es debido al intento de ayudar al usuario a realizar los test de un modo más sencillo y de conseguir que las pruebas sean lo más eficientes posibles. Como ya se ha dicho antes, para ejecutar una acción sobre algún control es necesario localizar primero la ventana que lo contiene, y después buscar entre los controles de dicha ventana hasta encontrar el deseado para finalmente ejecutar la acción. Según esto cada vez que se quiera ejecutar alguna acción se debe indicar en qué ventana está el control, por qué criterio buscar la ventana, cual es el atributo que identifica el control (es decir, el criterio de búsqueda) y los valores para ambos criterios. Además se puede pasar un numero que expresa en segundos el tiempo máximo de búsqueda, en caso de no pasarlo se utilizara el de por defecto. De esta manera el usuario puede realizar la búsqueda de los controles por el criterio que más le convenga en cada momento, *AutomationId, Nombre, Contenido, Clase del objeto,* etc.

¿Pero y que es lo que hace de todo esto que las pruebas sean más eficientes? Pues que el usuario puede no indicar la ventana si el control a utilizar esta sobre la misma que el anterior, así como tampoco debe indicar el control si desea ejecutar acciones consecutivas sobre el mismo control. De este modo vamos ahorrando una o dos búsquedas por cada acción lo que supone que si en la batería ejecutamos 100 acciones podemos llegar a ahorrar 198 búsquedas, ya que las dos primeras serán obligatorias.

Para poder realizar esto se utiliza la clase estática *TestParameter* accesible por todos los controles. En ella se va almacenando la ventana de trabajo actual cada vez que el usuario hace una nueva búsqueda. Esta clase además carga los parámetros configurables, los inicializa para la ejecución del test y define las constantes necesarias por la herramienta, como son las constantes de búsqueda, etc. Los parámetros configurables son el tiempo máximo de búsqueda, si se desea contabilizar el tiempo de cada acción para después dejarlo reflejado en el fichero log que contendrá los resultados de las acciones ejecutadas, la ruta donde dejar dicho fichero y la ruta donde encontrar la hoja de transformación para convertir el fichero de log en formato xml a un documento html o cualquier otro que se desee, sin más que modificar la hoja de transformación generando un fichero mucho más fácil de leer por parte del usuario.

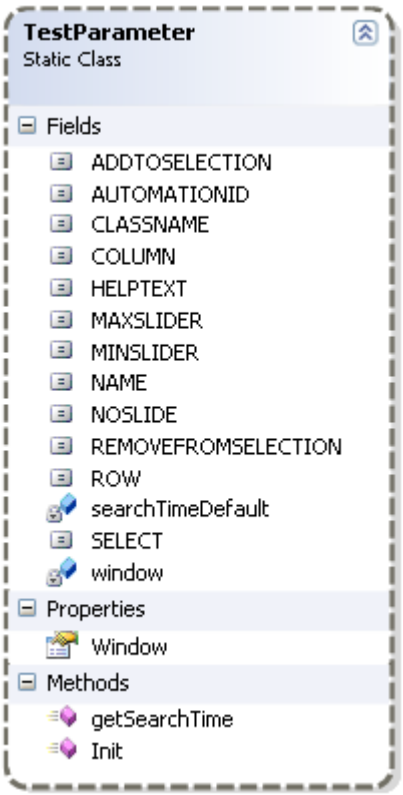


Figura 11.23. *Clases estática TestParameter.*

11.2.1 Framework UI Automation

Es el nuevo *framework* de accesibilidad de Microsoft Windows ®, disponible en todos los sistemas operativos que soporten *Windows Presentation Foundation ® (WPF)*, se encuentra por tanto incluido dentro del *.Net Framework 3.0*. Este *framework* tiene una gran importancia en el desarrollo e implementación de este proyecto, pues gracias a él la herramienta *X-Aute* puede ejecutar las acciones automáticas sobre los controles de interfaz sin necesidad que el usuario interactúe con ellos.

En este punto entran en juego los llamados patrones del control, en inglés *Control Patterns*, que son los distintos conjuntos de operaciones que podemos ejecutar sobre cada control. Las agrupaciones en torno a la funcionalidad que se han realizado en el desarrollo del sistema y han sido mostradas en el modelo de éste, están basadas en que tales grupos de controles soportan el mismo *Pattern*. Por ejemplo todos los controles del grupo *InvokeControl* soportan el *Pattern Invoke* por lo que a todos ellos se les puede realizar una *invocación* automática.

A continuación se realiza una detallada descripción de esta interesante librería mostrando su utilidad como *framework* para la codificación de los escenarios de testeo automatizados.

UI Automation provee un modelo de objetos unificado que hace posible que todos los *framework de interfaz de usuario* expongan funcionalidades complejas y ricas de un modo accesible y sencillo para la automatización.

Para realizar una herramienta útil de testo automatizado de *UI Automation* es necesario un proveedor (*UI Automation Provider*) y un cliente (*UI Automation Client*). Los *UI Automation Provider*s son aplicaciones basadas en los controles del sistema operativo Microsoft Windows ®, mientras que los *UI Automation Clients* incluyen scripts automatizados de prueba y aplicaciones de asistencia de la tecnología, como en este caso *UISpy*.

Por tanto el *UI Automation Provider* será la *aplicación WPF* que se desea testear y el *UI Automation Client* la herramientas *X-Aute,* junto con la aplicación *UISpy*, explicada más abajo.

11.2.1.1 UI Automation Provider

Para que las pruebas de una interfaz de usuario puedan ser automatizadas, el desarrollador de una aplicación o control personalizado debe fijarse en cuáles serán las acciones que el usuario final podrá realizar con el/los objeto/s de interfaz utilizando la interacción estándar mediante teclado y ratón.

Una vez que estas acciones clave han sido identificadas, el correspondiente *UI Automation Control Pattern* (es decir, los patrones del control que reflejan la funcionalidad y el comportamiento del elemento de interfaz de usuario) debe ser implementado en el control. Por ejemplo, la interacción del usuario con un *ComboBox* normalmente implica la expansión y el colapso del *ComboBox* para ocultar o mostrar la lista de valores, seleccionar un valor de la lista, o añadir un nuevo valor a través del teclado.

Los *Control Patterns* han sido desarrollados para representar los comportamientos comunes de los controles.

11.2.1.2 UI Automation Client

El objetivo para muchas herramientas de testeo automatizado es la manipulación consistente y repetible de la interfaz de usuario gráfica.

Una complicación que surge de las aplicaciones automatizadas es la dificultad de sincronizar una prueba con un objetivo dinámico. Por ejemplo, un control *ListBox*, como el contenido en el Administrador de tareas de Windows, que muestra una lista de las aplicaciones que están en ejecución actualmente. Como los elementos de la lista son actualizados dinámicamente desde fuera del control de la aplicación de test, tratar de repetir la selección de un elemento específico en el *ListBox* sin inconsistencias es imposible.

El ***acceso mediante programación*** ofrece la posibilidad de imitar, a través de código, toda la interacción expuesta por el tradicional ratón y teclado. *UI Automation* permite el acceso mediante programación a través de cinco componentes:

- El árbol de *UI Automation* facilita la navegación a través de la estructura de la interfaz de usuario. Se construye a partir de la recopilación de los manejadores de ventana (hWnd).
- Los *Automation Elements* son los componentes individuales de la interfaz de usuario, a menudo pueden ser más granular que un hWnd.
- Las *Automation Properties* ofrecen información específica sobre los elementos de la interfaz.
- Los *Control Patterns* definen aspectos particulares de la funcionalidad de un control, que puede consistir en propiedades, métodos, eventos, y la estructura de la información.
- Los *Automation Events* proporcionan notificaciones de eventos e información.

La capacidad de identificar para posteriormente poder localizar cualquier control dentro de la interfaz de usuario proporciona la base para las herramientas de pruebas automatizadas. Existen diversas *UI Automation Properties* que son utilizadas por los clientes y proveedores en este aspecto.

- ***AutomationID***: Identifica de forma única un elemento de automatización. La propiedad *AutomationId* es invariante y por tanto localizable, a diferencia de la propiedad *NameProperty*, si un producto se entrega en varios idiomas.
- ***ControlType***: Identifica el tipo de control representado por un *Automation Element*. Alguna información importante se puede inferir a partir del conocimiento del tipo de control.
- ***NameProperty***: Es una cadena de texto que explica o identifica a un control. Debe utilizarse con precaución ya que su valor puede variar y por tanto el control no será localizado.

11.2.1.3 Utilizando el framework UI Automation en una Aplicación de Test

Agregar las referencias al framework ***UIAutomation***.	Las dll's necesarias para los clientes *UI Automation* son: ♦ ***UIAutomationClient.dll***: Proporciona acceso a la interfaz del API *UI Automation Client-side*. ♦ ***UIAutomationClientSideProvider.dll***: Proporciona las habilidades necesarias para automatizar controles Win32. ♦ ***UIAutomationTypes.dll***: Proporciona acceso a los tipos específicos definidos en el *framework UI Automation*.
Agregar el namespace ***System.Windows.Automation***	Este namespace contiene todo lo que los *UI Automation Clients* necesitan para usar las capacidades del *framework UI Automation* a excepción de la manipulación de texto.
Agregar el namespace ***System.Windows.Automation.Text***	Este namespace contiene todo lo que los UI Automation Clients necesitan para usar las capacidades de manipulación de texto del framework UI Automation.

Encontrar los controles en que se esté interesado	Los scripts de test automatizados han de localizar en el árbol de UI Automation los Automation Elements que representan los controles que interesan al usuario. Existen múltiples maneras de obtener dichos Automation Elements: ♦ Realizar búsquedas sobre la interfaz de usuario indicándole una o más condiciones. En este tipo de localizaciones es donde se utiliza típicamente el *AutomationIdProperty*. ♦ Utilizar la clase *TreeWalker* para recorrer todo el árbol de *UI Automation*. ♦ Utilizar el hWnd del control. ♦ Utilizar la localización en pantalla, como la ubicación del cursor del raton.
Obtener los Control Patterns	Después de localizar los controles, los scripts de prueba automatizados obtienen los *Control Patterns* de interés de los *Automation Elements* localizados. Por ejemplo, el *Pattern InvokePattern* típico de la funcionalidad de botón.
Automatizar la Interfaz de Usuario Gráfica	Ahora, los script de test automatizados pueden controlar la interfaz de usuario de interés utilizando la información y la funcionalidad expuesta por el Control Pattern.

Tabla 11.1. *Utilización del framework UI Automation para una Aplicación de Test.*

11.2.1.4 Herramientas relacionadas

UISpy es una aplicación con interfaz de usuario gráfica que puede ser utilizada para recoger la información de la interfaz de usuario. Está incluida en el *Windows Software Development Kit ® (Windows SDK ®)*.

11.2.1.5 Más detalles acerca de los Control Patterns

Los *Patterns* proporcionan una manera de categorizar y exponer la funcionalidad de un control independientemente de su tipo o apariencia.

El *framework UI Automation* utiliza los *Patterns* para representar comportamientos de control comunes. Por ejemplo, se usa el patrón de control Invoke para los controles que se pueden invocar (como los botones) y el patrón de control Scroll para los controles que tienen barras de

desplazamiento (como cuadros de lista, vistas de lista o cuadros combinados). Dado que cada patrón de control representa una funcionalidad independiente, se pueden combinar para describir toda la funcionalidad que admite un control determinado.

Los *Control Patterns* admiten los métodos, las propiedades, los eventos y las relaciones que son necesarios para definir una parte discreta de la funcionalidad que está disponible en un control.

La relación entre un *Automation Element* de la interfaz de usuario y su elemento primario, sus elementos secundarios y sus elementos relacionados describe la estructura del elemento en el árbol de *UI Automation*.

Los métodos permiten a los *UI Automation Clients* manipular el control, las propiedades y los eventos proporcionan información acerca de la funcionalidad del *Pattern*, así como información acerca del estado del control.

Los *Control Patterns* se relacionan con la interfaz de usuario como las interfaces se relacionan con los objetos COM. En COM, se puede consultar un objeto para averiguar qué interfaces admite y, a continuación, usar esas interfaces para tener acceso a funcionalidad. En el *framework UI Automation*, los clientes pueden preguntar a un control qué *Patterns* admite y, a continuación, interactuar con el control a través de las propiedades, los métodos, los eventos y las estructuras expuestos por dichos *Patterns*. Por ejemplo, para un cuadro de edición de varias líneas, los *UI Automation Providers* implementan la interfaz *IScrollProvider*. Cuando un cliente sabe que el *Automation Element* admite el *Pattern ScrollPattern*, puede usar las propiedades, los métodos y los eventos expuestos por ese *Pattern* para manipular el control o tener acceso a información sobre el mismo.

Los *UI Automation Providers* implementan *Control Patterns* para exponer el comportamiento adecuado de una parte específica de la funcionalidad que admite el control.

Los *UI Automation Clients* obtienen acceso a los métodos y las propiedades de las clases de los *Patterns* y los usan para obtener información acerca de la interfaz de usuario o para manipularla. Las clases de los *Control Patterns* se encuentran en el espacio de nombres ***System.Windows.Automation*** (por ejemplo, *InvokePattern* y *SelectionPattern*).

En la siguiente tabla se describen los *Control Patterns* del framework *UI Automation*. Se muestran también las clases utilizadas por *UI Automation Clients* para tener acceso a los *Patterns*, así como también las interfaces utilizadas por los *UI Automation Providers* para implementarlos.

Clase del Pattern	*Interfaz del UI Automation Provider*	*Descripción*
DockPattern	*IDockProvider*	Se usa para controles que se pueden acoplar en un contenedor de acoplamiento. Por ejemplo, barras de herramientas o paletas de herramientas.
ExpandCollapsePattern	*IExpandCollapseProvider*	Se usa para controles que se pueden expandir o contraer. Por ejemplo, los elementos de menú de una aplicación, como el menú Archivo.
GridPattern	*IGridProvider*	Se usa para controles que admiten funcionalidad de cuadrícula, como la modificación del tamaño y el desplazamiento a una celda especificada. Por ejemplo, la vista de iconos grandes del Explorador de Windows o las tablas simples sin encabezados de Microsoft Word.
GridItemPattern	*IGridItemProvider*	Se usa para controles que tienen celdas dentro de cuadrículas. Cada una de las celdas debería admitir el patrón GridItem. Por ejemplo, cada celda de la vista de detalles del Microsoft Windows Explorer.
InvokePattern	*IInvokeProvider*	Se usa para controles que se pueden invocar, como un botón.
MultipleViewPattern	*IMultipleViewProvider*	Se usa para controles que pueden cambiar entre varias representaciones del mismo conjunto de información, datos o elementos secundarios. Por ejemplo, un control de vista de lista donde los datos están disponibles como imágenes en miniatura, en mosaico, como iconos, como una lista o con detalles.
RangeValuePattern	*IRangeValueProvider*	Se usa para controles que tienen un intervalo de valores que se puede aplicar al control. Por ejemplo, un control de número que contiene años podría tener un intervalo de 1900 a 2010, mientras que otro control de número que presenta meses tendría un intervalo de 1 a 12.
ScrollPattern	*IScrollProvider*	Se usa para controles que se pueden desplazar. Por ejemplo, un control que tiene barras de desplazamiento que están activas cuando hay más información de la que se puede mostrar en su área visible.

ScrollItemPattern	*IScrollItemProvider*	Se usa para controles que tienen elementos individuales en una lista que se desplaza. Por ejemplo, un control de lista que tiene elementos individuales en la lista de desplazamiento, como un control de cuadro combinado.
SelectionPattern	*ISelectionProvider*	Se usa para controles contenedores de selecciones. Por ejemplo, cuadros de lista y cuadros combinados.
SelectionItemPattern	*ISelectionItemProvider*	Se usa para elementos individuales en controles contenedores de selecciones, como cuadros de lista y cuadros combinados.
SelectionPattern	*ISelectionProvider*	Se usa para controles contenedores de selecciones. Por ejemplo, cuadros de lista y cuadros combinados.
SelectionItemPattern	*ISelectionItemProvider*	Se usa para elementos individuales en controles contenedores de selecciones, como cuadros de lista y cuadros combinados.
TablePattern	*ITableProvider*	Se usa para controles que tienen una cuadrícula e información de encabezado. Por ejemplo, hojas de cálculo de Microsoft Excel.
TableItemPattern	*ITableItemProvider*	Se usa para los elementos de una tabla.
TextPattern	*ITextProvider*	Se usa para controles de edición y documentos que exponen información textual.
TogglePattern	*IToggleProvider*	Se usa para controles donde se puede alternar el estado. Por ejemplo, casillas y elementos de menú que pueden activarse.
TransformPattern	*ITransformProvider*	Se usa para controles que se pueden cambiar de tamaño, mover y girar. Los usos típicos del patrón de control Transform se encuentran en diseñadores, formularios, editores gráficos y aplicaciones de dibujo.

ValuePattern	*IValueProvider*	Permite que los clientes obtengan o establezcan un valor en controles que no admiten un intervalo de valores. Por ejemplo, un selector de fecha y hora.
WindowPattern	*IWindowProvider*	Expone información específica de las ventanas, un concepto fundamental para el sistema operativo Microsoft Windows. Algunos ejemplos de controles que son ventanas son las ventanas de la aplicación de nivel superior (Microsoft Word, Microsoft Windows Explorer, etc.), las ventanas secundarias de la interfaz de múltiples documentos (MDI) y los cuadros de diálogo.

Tabla 11.2. *Clasificación de los Control Patterns.*

11.3 EJECUCIÓN Y CREACIÓN DE SCRIPTS

En este apartado se va a contar la otra parte de la herramienta, que la completa y la convierte es una aplicación útil para poder crear los test automatizados que posteriormente se van a ejecutar sobre una interfaz de usuario gráfica.

El usuario puede optar por desarrollar los test del modo que prefiera, pero *X-Aute* se ha desarrollado pensando que para escribir los scripts se utilizarán las dos plantillas que se han creado y que generan las baterías de test con las siguientes características.

Todos los ficheros de la batería definirán y complementarán la misma clase, ya que se han utilizado las clases parciales, lo que permite ofrecer mayor facilidad para el usuario en la creación de los tests de pruebas, a continuación se irá viendo por qué. El nombre de la clase parcial será *TestBattery* y el *namespace* será el nombre que se le asigne al proyecto y que inicialmente estará formado por los ficheros *TestInit* y *TestRunner,* que son los ficheros que agrega la plantilla-proyecto de las Baterías de Test.

En el fichero *TestInit* se declaran e inicializan todas las constantes como los criterios de búsqueda, etc. necesarias para la creación y posterior ejecución de los test así como también se declaran e inicializan cada uno de los objetos de los controles cuyas acciones el usuario podrá automatizar. El método *Init*, definido en el fichero *TestInit* inicializa, además de lo ya comentado, los valores necesarios para la creación del fichero de log con las trazas de la ejecución. En este

fichero existirán dos métodos que serán modificables y en los que se indicará cual es la aplicación que se desea testear para poder iniciarla al comenzar el test y cerrarla al finalizar.

El fichero *TestRunner* tan solo contiene el método Main de la clase y dentro de éste una llamada al método *Init* definido en el fichero *TestInit*. Debajo de dicha llamada a la función *Init* se incorporarán las invocaciones a los casos de test que se vayan agregando.

Los casos de test se insertarán utilizando la plantilla-ítem de los Casos de Test. Este *template* agregará un fichero por cada caso de test que se inserte al proyecto. Cada uno de esos ficheros formará parte de la clase parcial *TestBattery*, es decir será la misma clase que definen los ficheros que la plantilla de la batería inserta al proyecto. Cada uno de estos ficheros que se agregan cuando se añade un caso de test al proyecto, no tendrá en inicio más que la definición de un método con el nombre que se le haya dado al caso y dos llamadas a funciones relacionadas con las trazas y encargadas de empezar y finalizar, respectivamente, la escritura de las acciones dentro del caso de test. Entre tales llamadas se intercalarán las acciones que el usuario quiere automatizar para realizar el testeo de la interfaz *WPF*. En la parte superior del caso de test habrá una zona de comentarios que el usuario desarrollador del script de prueba debe rellenar con el objetivo de dejar los casos bien documentados con información como *Autor, Nombre del Test, Fecha, Componente Testeado,* etc.

```
/*****************************************************************************
******************************************************************************

 TEST NAME              :
 COMPONENT              :
 AUTHOR                 :
 DATE                   :
 LANGUAGE               : C#
 OPERATING SYSTEM       : Microsoft Windows

 PURPOSE                :

 PREVIOUS STATE         :

 SUCCESSFUL EXIT STATE  :

 HISTORY

    <Ver>   <Programer>     <Date>      <Comment>


******************************************************************************
******************************************************************************/

using System;
using System.Collections.Generic;
using System.Text;

namespace SampleTestBattery
{
    partial class TestBattery
    {
        static void SampleTestCase()
        {
            WriteTestCase("SampleTestCase");

            // Añadir aqui las acciones a realizar.

            WriteTestCaseResult();
        }
    }
}
```

Figura 11.24. *Código de la plantilla TestCase.*

11.3.1 Ejemplo de creación de una batería de pruebas

Para crear los test de un modo rápido se han creado las dos plantillas ya comentadas. La primera permite definir de un modo muy sencillo una nueva Batería de Test. Para ello tan sólo hay que crear en el entorno de desarrollo Visual Studio 2005® un nuevo proyecto seleccionando como tipo de proyecto la plantilla "*TestBatteryTemplate*" e indicarle un nombre que la misma plantilla se encarga de escribir en los lugares que sea necesario.

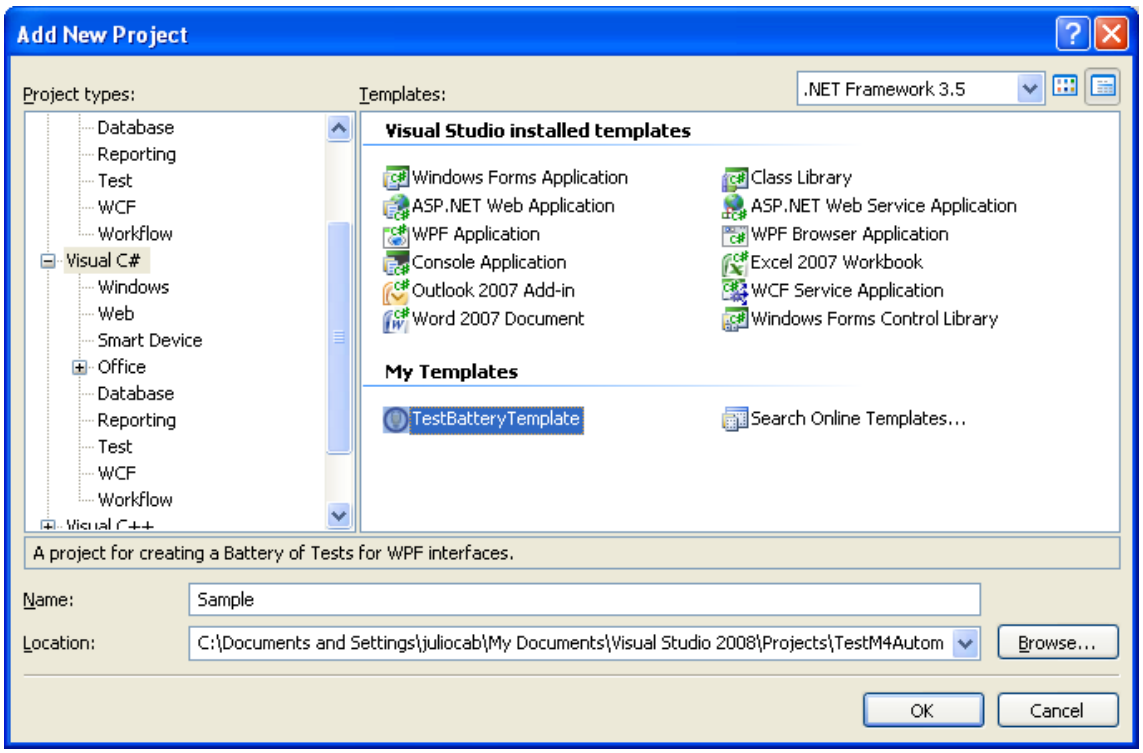


Figura 11.25. *Plantilla de las Baterías de Test.*

Esto creará un proyecto que contendrá dos ficheros, en los cuales el usuario no debe modificar nada a excepción de añadir en el método *Main* del fichero *TestRunner* la invocación a los casos de test que vaya añadiendo, algo tan simple como escribir en dicho fichero los nombres de los casos de test agregados con formato de método C#, es decir el nombre seguido de "()".

```
using System;
using System.Collections.Generic;
using System.Text;

namespace SampleTestBattery
{
    partial class TestBattery
    {
        [STAThread]
        public static void Main(string[] args)
        {
            Init();
            // Add here the Test Cases
            SampleTestCase();
            SampleTestCase2();

            ...

            End();
        }
    }
}
```

Figura 11.26. *Ejemplo de código del fichero TestRunner.*

La otra plantilla ha sido definida para agregar los casos de test a la batería. Para ello habrá que añadir un nuevo elemento a la batería mediante "Botón derecho → Add → New Item…", seleccionar la plantilla "*TestCaseTemplate*" e indicarle un nombre.

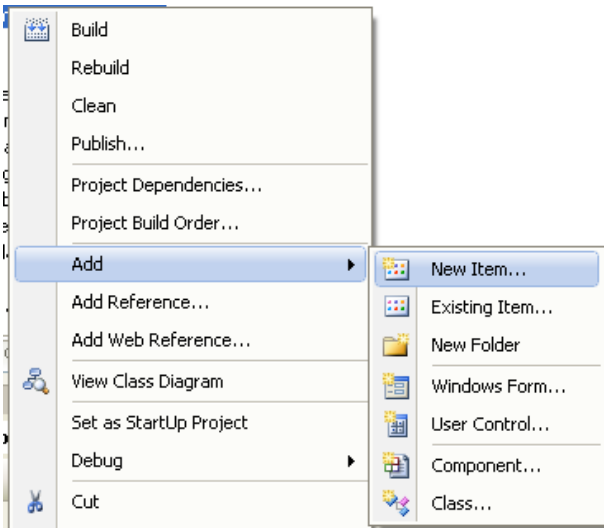


Figura 11.27. *Ejemplo de inserción de Caso de Test.*

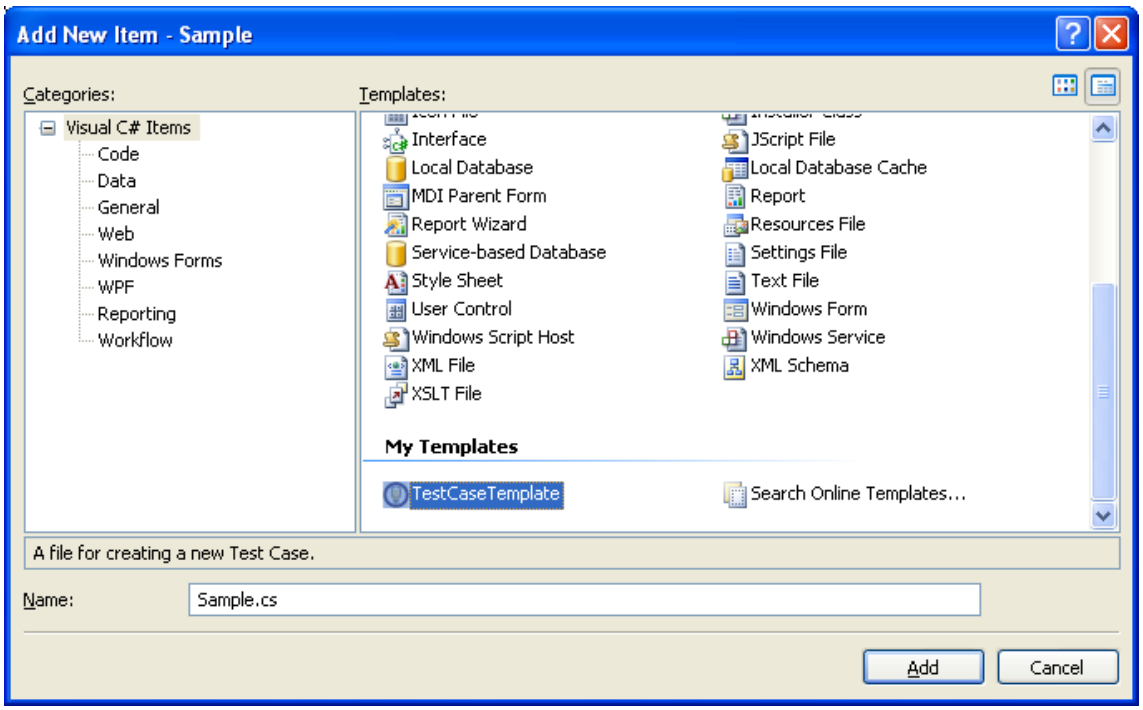


Figura 11.28. *Plantilla de los Casos de Test.*

Esta propia plantilla se encarga de poner el nombre donde es necesario y el usuario tan sólo tendrá que empezar a escribir las acciones que desea se ejecuten automáticamente.

Como los ficheros que añaden los casos de test definen la misma clase parcial que la plantilla de la batería, en donde se definen e instancian los objetos, el usuario puede invocar dichos objetos sin tener que crearlos ni instanciarlos.

Todo el código de la herramienta ha sido documentado y se ha generado un fichero XML de documentación que contiene la descripción de cada clase, atributo, método, parámetros de entrada de los métodos y valores de retorno. Todo esto facilita aún más el trabajo del usuario pues según va escribiendo se muestra toda la información relativa a los controles y a sus acciones. Y junto con el *IntelliSense* que ofrece el entorno de desarrollo *Visual Studio 2005*® la escritura de scripts se convierte es una tarea muy sencilla.

```
/// <summary>
/// Lanza una petición para activar un control e iniciar su acción.
/// Si no se especifica ventana se establece como tal la ventana de trabajo definida actualmente,
cuando se especifica se guarda como ventana de trabajo.
/// Si tampoco se indica el control se utiliza el Control establecido anteriormente.
/// Si se desea utilizar el tiempo de búsqueda por defecto no se ha de pasar el último parámetro.
/// </summary>
/// <param name="window">El valor de búsqueda para la ventana que contiene el control que se quiere
automatizar.</param>
/// <param name="windowCriteria">Especifica el criterio mediante el cual se ha de buscar la
ventana.</param>
/// <param name="control">El valor de búsqueda para el control.</param>
/// <param name="controlCriteria">Especifica el criterio mediante el cual se ha de buscar el
control.</param>
/// <param name="searchTime">Tiempo de búsqueda máximo, expresado en segundos. Si se desea utilizar el
tiempo por defecto no se debe pasar nada.</param>
public void Invoke(string window, int windowCriteria, string control, int controlCriteria, params int[]
searchTime)
{
            . . .
}
```

Figura 11.29. *Ejemplo de clase documentada.*

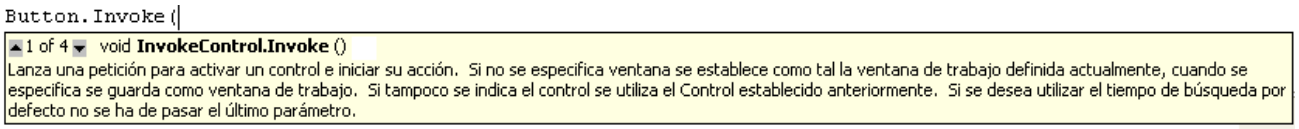


Figura 11.30. *Ejemplo de documentación mostrada al usuario (I).*

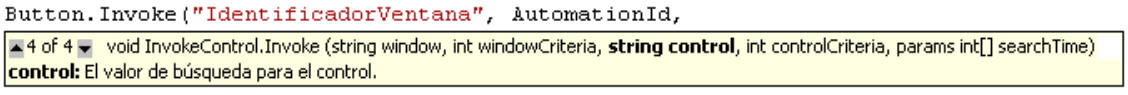


Figura 11.31. *Ejemplo de documentación mostrada al usuario (II).*

11.3.2 Resultados de ejecución de los test

La ejecución de estos test genera unos resultados que se almacenan en un fichero log con formato *XML*. Se ha elegido guardar los datos en *XML* puesto que es un formato muy adecuado para almacenar este tipo de documentos además de que mediante hojas de trasformación se pueden convertir en el formato que el usuario desee.

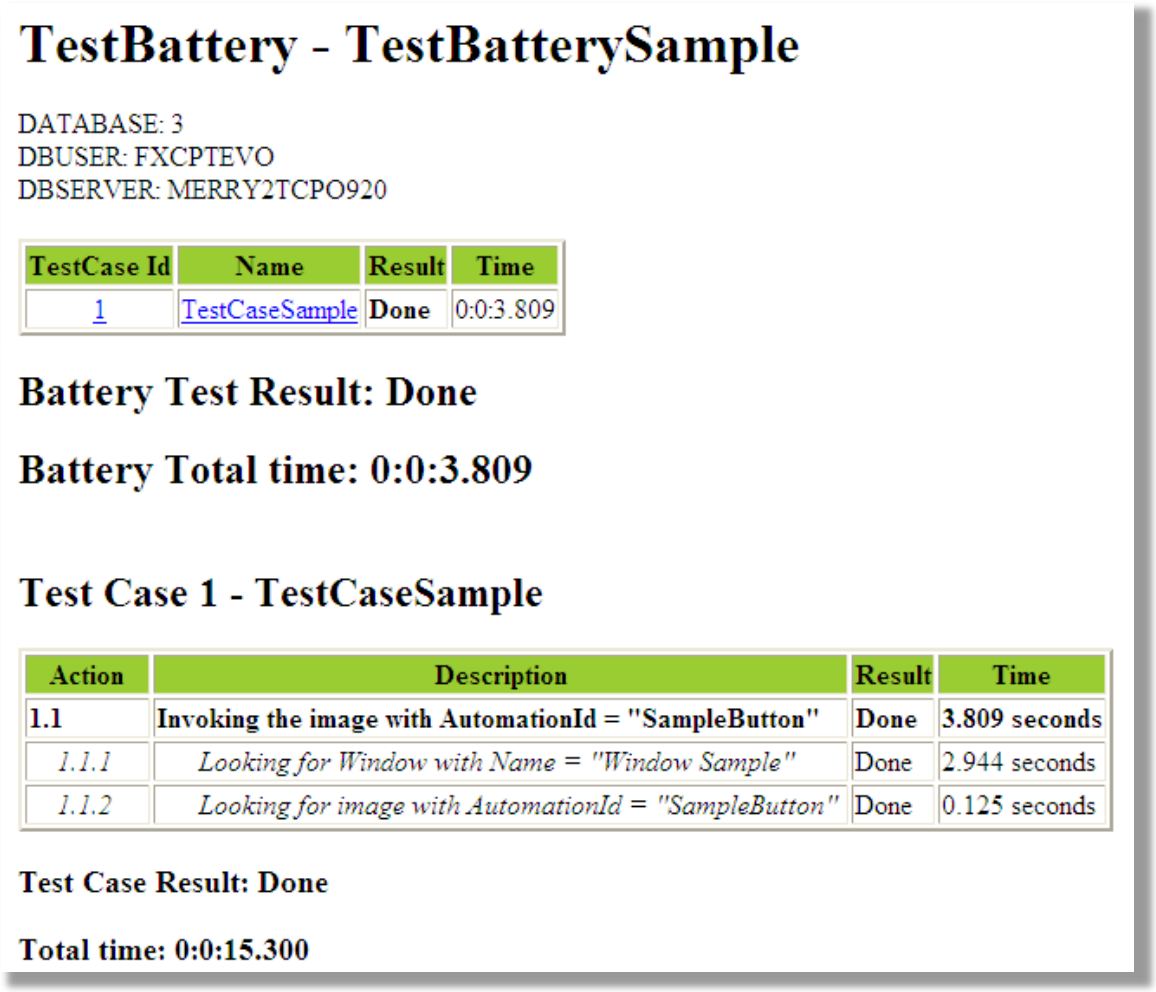
En este fichero de trazas se almacena un primer elemento, elemento raíz de todo el documento *XML*, *TestBattery* que hace referencia a la batería completa, con sus propiedad nombre. Dentro de este se van almacenando los elementos *TestCase*, que hacen referencia a cada uno de los casos de test, con las propiedades nombre e identificador. Los elementos *Action*, que son las acciones que el usuario ha escrito en el script de test, se almacenan dentro de los casos de test como hijos de primer nivel mientras que las sub-acciones, *RaisedAction*, que han sido provocadas con motivo de la ejecución de la instrucción dada por el usuario se guardan anidadas en la acción que la provoco. Véase un ejemplo, el usuario quiere pulsar un botón, esta acción provoca dos sub-acciones que son la búsqueda de la ventana y la búsqueda del control para poder finalmente pulsar dicho botón.

```
- <TestBattery Name="TestBatterySample">
    <DBSERVER>MERRY2TCPO920</DBSERVER>
    <DBUSER>FXCPTEVO</DBUSER>
    <DATABASE>3</DATABASE>
  - <TestCase Name="TestCaseSample" ID="1">
    - <Action ID="1.1">
        <Description>Invoking the Button with AutomationId = "SampleButton"</Description>
      - <RaisedAction ID="1.1.1">
          <Description>Looking for Window with Name = "Window Sample"</Description>
          <Result>Done</Result>
          <Time>2.944 seconds</Time>
        </RaisedAction>
      - <RaisedAction ID="1.1.2">
          <Description>Looking for Button with AutomationId = "SampleButton"</Description>
          <Result>Done</Result>
          <Time>0.125 seconds</Time>
        </RaisedAction>
        <Result>Done</Result>
        <Time>3.809 seconds</Time>
      </Action>
    </TestCase>
    <BatteryResult>Done</BatteryResult>
    <BatteryTime>0:0:3.809</BatteryTime>
  </TestBattery>
```

Figura 11.32. *Ejemplo de fichero log con el resultado de la ejecución de la Batería de Test.*

El comportamiento que la herramienta tiene si la misma batería ya ha sido ejecutada y encuentra el anterior fichero de log es el de interpretar el contenido de dicho fichero. Mantiene los casos de test ejecutados anteriormente y agrega los casos de test que se ejecuten en esta iteración actualizando el tiempo total de la batería y su resultado final.

Dentro del proceso de desarrollo de *X-Aute* se ha creado una hoja *XSLT* que transforma el *XML* a *HTML* haciendo la consulta de los resultados mucho más legible para el usuario. Dicha hoja puede modificarse para ajustar la conversión a los gustos del usuario.



TestBattery - TestBatterySample

DATABASE: 3
DBUSER: FXCPTEVO
DBSERVER: MERRY2TCPO920

TestCase Id	Name	Result	Time
1	TestCaseSample	Done	0:0:3.809

Battery Test Result: Done

Battery Total time: 0:0:3.809

Test Case 1 - TestCaseSample

Action	Description	Result	Time
1.1	Invoking the image with AutomationId = "SampleButton"	Done	3.809 seconds
1.1.1	*Looking for Window with Name = "Window Sample"*	Done	2.944 seconds
1.1.2	*Looking for image with AutomationId = "SampleButton"*	Done	0.125 seconds

Test Case Result: Done

Total time: 0:0:15.300

Figura 11.33. *Ejemplo de fichero log en formato HTML.*

La hoja *XSLT* actual genera un *HTML* que muestra una tabla con el titulo de cada uno de los casos de test ejecutados y el resultado final de su ejecución completa. Cada título será un link al propio caso con los detalles de cada una de las acciones ejecutadas. Después de la tabla de títulos de casos se muestra el resultado total de la batería.

Si en el fichero de configuración está indicado, se mostrara también el tiempo de ejecución de cada acción (incluyendo sus sub-acciones), el tiempo total de cada caso y el tiempo total de la batería.

SECCIÓN
IV
CONCLUSIONES Y LÍNEAS FUTURAS

12. CONCLUSIONES

Los Sistemas de Información Corporativos y las Tecnologías de la Información han cambiado la forma en que operan las organizaciones actuales. A través de su uso se logran importantes mejoras como ventajas competitivas o reducir la ventaja de los competidores. Hoy en día, las organizaciones se apoyan cada vez más en sistemas de información eficientes para poder hacerse un hueco en los mercados internacionales y mejorar la calidad de sus sistemas productivos. Es por tanto necesario que tales productos software sean de calidad.

Al igual que todas las aplicaciones informáticas, los Sistemas de Información Corporativos van evolucionando y son cada vez más sofisticados. Dicha sofisticación está relacionada con las funcionalidades y las operaciones capaces de realizar por la herramienta, pues este tipo de aplicaciones tienen como uno de sus objetivos ofrecer un uso sencillo. Para lograr este objetivo se emplean las interfaces de usuario gráficas, que permiten interactuar con la herramienta de un modo muy simple e intuitivo. Con la aparición de la nueva tecnología *WPF* es posible desarrollar interfaces de usuario gráficas muy ricas y potentes tanto en funcionalidades como en los nuevos aspectos visuales. Por sus características, los Sistemas de Información Corporativos comienzan a demandar este tipo de interfaces gráficas.

Para asegurar que los Sistemas de Información desarrollados son aplicaciones software de calidad, eficientes y eficaces es necesario realizar un proceso de pruebas en toda la aplicación, lo que, por supuesto, también incluye las nuevas y potentes interfaces *WPF*.

El testeo automatizado es una de las mejores alternativas para que las organizaciones realicen sus pruebas consiguiendo ahorrar gran cantidad de tiempo y recursos y produciendo software de mayor calidad más rápido de lo que lo harían utilizando tan sólo el testeo manual. Más aún cuando se habla del testeo de las interfaces gráficas que son la parte del software que mas variaciones sufre, y por tanto la parte en la que hay que realizar y repetir mayor número de pruebas.

Se ha diseñado una solución que posibilita un sencillo y eficaz testeo de las interfaces de usuario gráficas *WPF*. La herramienta permite crear los scripts de prueba automatizados sin requerir que el usuario encargado de dicha tarea posea unos conocimientos técnicos elevados en ninguna

materia concreta, pudiéndose utilizar dos plantillas elaboradas para tal fin y que facilitan mucho esta creación. Además la herramienta es capaz de ejecutar dichos scripts manipulando automáticamente y sin interacción alguna por parte del usuario, eliminando así las probabilidades de sus errores, la interfaz que se pretenda someter al proceso de pruebas.

En el diseño e implementación de la herramienta se ha tenido en cuenta el valor del resultado de cada una de las acciones ejecutadas por la batería de pruebas automatizada, pues si este no se deja reflejado la ejecución del test no tiene sentido puesto que no se puede extraer ninguna conclusión de ella. Por esto la herramienta genera un documento *XML,* que más tarde transforma a *HTML* o cualquier otro formato que le indique la hoja de transformación *XSLT,* en el que documenta cada acción ejecutada y su resultado así como también algunas otras características.

A lo largo de todo el ciclo de vida del proyecto se han aplicado los estándares de Ingeniería del Software publicados por la Agencia Espacial Europea, lo que garantiza la calidad y la obtención de buenos resultados en el desarrollo por la utilización de las mejores prácticas que establecen.

Desde el punto de vista académico se ha realizado un trabajo integrador que ha abarcado desde las tareas de investigación para la redacción del estado del arte hasta el desarrollo de una herramienta software pasando por las necesarias labores de la gestión del proyecto de desarrollo, captura de requisitos, etc. definidos en el ciclo de vida del mismo.

Por último, habría que destacar que la relevancia y profundidad de las aportaciones de este *Proyecto Fin de Carrera* están avaladas por la aplicación de sus estudios y de su producto final en el entorno de desarrollo de una empresa líder en el desarrollo y comercialización de software paquetizado de ámbito empresarial.

13. LÍNEAS FUTURAS

A pesar de los frutos obtenidos tras la realización de este *Proyecto Fin de Carrera* y llegados a este punto se puede plantear un conjunto de posibles líneas de trabajo futuras que permitan extender y evolucionar los logros alcanzados.

Una primera línea de trabajo consistiría en el desarrollo de una herramienta grabadora que "*escuche*" las acciones en la interacción del usuario con la interfaz gráfica y vaya generando automáticamente un script con cada una de las acciones que el usuario realiza. Dicho script generado automáticamente sería similar a los que se crean ahora mismo a mano con la herramienta, es decir invocaría la funcionalidad desarrollada en este proyecto.

En la actualidad el problema que se le presenta a este tipo de herramientas es la identificación y localización de los controles. Ya que existen aplicaciones en las que los controles no tienen identificadores, estos no son unívocos o son controles dinámicos creados en tiempo de ejecución, y no tienen ningún otro tipo de propiedad que permita localizarlos. Esta sería otra línea de trabajo en la que se debería desarrollar una herramienta capaz de comprender cada interfaz y, recorriendo el árbol visual o lógico de la interfaz, ir asignando un identificador o valor a alguna propiedad que permita identificar y localizar unívocamente cada uno de los controles de la *GUI*.

Una línea de trabajo situada en la parte opuesta al presente proyecto, pero con una gran relación entre ambos, sería la de extender, desde la parte del proveedor de la automatización de la interfaz de usuario (*UI Automation Provider*), es decir, desde la parte de la interfaz de usuario a testear, los comportamientos comunes que exporta cada control (*Control Patterns*). Para hacer que controles que, según lo definido por el estándar del *framework UI Automation,* no soportan algún *Pattern* pero sin embargo el desarrollador de la interfaz si les ha dotado de ese comportamiento, pueden exportar ese *Pattern* y poder después invocar automáticamente dicho comportamiento. Además se podría investigar la posibilidad de crear nuevos *Patterns* sin tener que limitarse a los que el *framework UI Automation* declara.

Por último se podría ampliar la herramienta desarrollada añadiendo las clases que modelen nuevos objetos o *Patterns* que vayan apareciendo.

SECCIÓN

V

ANEXOS

14. ANEXO: DOCUMENTOS DE LA ESA

14.1 DOCUMENTO DE REQUISITOS DE USUARIO

Este anexo contiene el Documento de Requisitos de Usuario *(DRU)* asociado al desarrollo de este proyecto. El documento es el producto de la primera fase del ciclo de desarrollo de software llamada *Fase de Definición de Requisitos de Usuario* o *Fase RU* (ver sección III, página 78, 79 y 80).

El *DRU* proporciona una definición de cuáles son los requisitos solicitados por el usuario siguiendo las directrices para la captura y especificación de requisitos de usuario propuestas por las guías para la aplicación de los estándares de Ingeniería de Software de la ESA (ESA Board for Software Standardisation and Control, 1991) y (ESA Board for Software Standardisation and Control, 1996).

Universidad Carlos III de Madrid

X-Aute

Herramienta para la Automatización de las pruebas en las Interfaces de Usuario Graficas desarrolladas con XAML

Documento de Requisitos de Usuario

Iteración:	2
Revisión:	0
Referencia:	DRU
Creación:	15/01/2008
Última modificación:	28/07/2008
Autor:	Ayllón Bonet, Julio César
Revisor:	Colomo Palacios, Ricardo

Este documento ha sido elaborado a partir de la traducción de las Plantillas CERN PSS-05. Las Plantillas CERN PSS-05 han sido elaboradas por el Programming Technology Group, ECP Division, CERN (El Laboratorio Europeo de Física de Partículas) y siguiendo los Estándares de Ingeniería de Software PSS-05 (ISBN 0-13-106568-8) definidos por el BSSC (Board for Software Standardization and Control) de la ESA (European Space Agency).

HOJA DE ESTADO DEL DOCUMENTO

TÍTULO DEL DOCUMENTO	**X-Aute** *Herramienta para la Automatización de las pruebas en las Interfaces de Usuario Graficas desarrolladas con XAML*
NÚMERO DE REFERENCIA DEL DOCUMENTO	*DRU*
RESPONSABLE DEL DOCUMENTO	*Julio César Ayllón Bonet*
REALIZADO POR	*Julio César Ayllón Bonet*

ITERACIÓN	REVISIÓN	FECHA	RAZÓN DE CAMBIO
1	0	15/01/2008	Versión inicial del documento
2	1	22/07/2008	Inicio segunda iteración del proyecto

REGISTRO DE CAMBIOS EN EL DOCUMENTO

Esta sección contiene las tablas que registran los cambios que se producen en el documento después de cada revisión del mismo. No se han registrado cambios desde el inicio de la segunda iteración del proyecto, por lo que este registro está vacío.

ÍNDICE GENERAL

1. Introducción

El presente documento contiene todos los requisitos de usuario necesarios para desarrollar la herramienta de testeo *X-Aute* que permitirá automatizar las pruebas de las interfaces de usuario gráficas desarrolladas mediante *XAML*.

Esta sección aborda los objetivos tanto de la herramienta desarrollada como de este documento en sí mismo. También recoge las definiciones y las referencias que aparecen en el resto del documento y una breve descripción del contenido de las secciones que lo conforman.

1.1 Propósito del documento

Este documento es el producto de la primera fase del ciclo de desarrollo de software llamada Fase de Definición de Requisitos de Usuario *(Fase RU)*. La *Fase RU* también recibe la denominación de fase de "definición de problema" dentro del ciclo de vida. En esta fase se transforma un conjunto de ideas sobre una tarea que se puede llevar a cabo con un determinado equipamiento informático en una definición de aquello que se espera del sistema se va a construir.

El DRU proporciona una definición de cuáles son los requisitos solicitados por el usuario siguiendo las directrices para la captura y especificación de requisitos de usuario propuestas por las guías para la aplicación de los estándares de Ingeniería de Software de la ESA (ESA Board for Software Standardisation and Control, 1996) y (ESA Board for Software Standardisation and Control, 1995).

1.2 Ámbito de la herramienta

X-Aute es una herramienta ideada para automatizar las pruebas de regresión y funcionales sobre las interfaces gráficas desarrolladas mediante *WPF/XAML*.

Desde el punto de vista del usuario final la utilización de la herramienta ha de ser lo más sencilla posib le teniendo en cuenta la complejidad que tiene la automatización de las pruebas *GUI*.

El usuario se creará uno o varios script de prueba en los cuales va declarando, de un modo muy sencillo e intuitivo, las acciones sobre los controles de la interfaz y que se pretende la herramienta ejecute automáticamente sobre la aplicación a testear – la interfaz gráfica *WPF*.

En ningún momento el usuario será consciente de la búsqueda de cada uno de los controles y la forma en que se accionan, ni tendrá que crear o inicializar objetos para poder hacer referencia a dichos controles siendo todo esto transparente para él.

La herramienta dejará un fichero de trazas con el resultado global de la ejecución de todos los casos de test y el desglose de cada uno con sus acciones correspondientes.

Las características específicas de la plataforma han sido determinadas por las particularidades, tanto exclusivas como derivadas de su integración, de las tres áreas del conocimiento que sirven como soporte a este proyecto y que son:

- La implantación de *Sistemas de Información* en las organizaciones.
- La utilización de *WPF* y *XAML* en la definición de *Interfaces de Usuario*.
- La *automatización* de las pruebas en las *Interfaces de Usuario Gráficas*.

1.3 Definiciones, acrónimos y abreviaturas

1.3.1 Definiciones

GUI Son las siglas inglesas de *Interfaz de Usuario Gráfica*. Programa software que gestiona la interacción con el usuario de manera gráfica mediante el uso de ventanas, iconos, controles comunes como menús, botones, y con la utilización del ratón y el teclado por parte del usuario.

1.3.2 Acrónimos

CERN Conseil Européen pour la Recherche Nucléaire. (Organización Europea para la Investigación Nuclear).

ESA European Space Agency (Agencia Espacial Europea).

RU Requisito de Usuario.

DRU Documento de Requisitos de Usuario.

SI Sistemas de Información.

IU Interfaces de Usuario.

WPF Windows Presentation Foundation.

XAML Extensible Application Markup Language.

1.3.3 Abreviaturas

No aplicable.

1.4 Referencias

ESA Board for Software Standardisation and Control. (1991). *ESA Software Engineering Standars* (Vol. 2). European Space Agency.

ESA Board for Software Standardisation and Control. (1996). *Guide to applying the ESA software engineering standards to small software projects.* European Space Agency.

Disponible en http://styx.esrin.esa.it/premfire/Docs/Bssc962.pdf

ESA Board for Software Standardisation and Control. (1995). Guide to the user requeriments definition phase.

Disponible en http://styx.esrin.esa.it/premfire/Docs/PSS0502.pdf

1.5 Estructura del documento

El presente documento está organizado siguiendo la estructura que se detalla a continuación:

La sección 2 proporciona los factores generales de los que dependen tanto la herramienta en desarrollo como sus requisitos. También establece las restricciones que delimitarán la labor del desarrollo del proyecto así como una descripción del modelo de casos de uso de la plataforma.

La sección 3 ofrece una descripción de los requisitos tanto funcionales como no funcionales.

Por último, la sección 4 contiene una enumeración de los requisitos de usuario capturados.

2. Descripción General

Esta sección ofrece una descripción general de la herramienta a desarrollar, estableciendo los requisitos y las restricciones de usuario más destacadas.

2.1 Perspectiva del producto

Los *Sistemas de Información* se han convertido en elementos fundamentales a la hora de tomar decisiones de cualquier tipo, estratégicas, tácticas, etc., dentro de las organizaciones. Los cambios introducidos por las nuevas tecnologías, *WPF* y *XAML*, y las nuevas capacidades que estas aportan a los *SI* utilizados en el entorno laboral hacen posible la adopción de *XAML* como estándar para el desarrollo de *IU* para las aplicaciones de los *SI* corporativos. Es por ello junto con la necesidad de un testeo eficaz del software lo que hace de la herramienta a desarrollar en el presente *Proyecto Fin de Carrera* una interesante y novedosa aplicación que sirva para asentar las bases de posteriores desarrollos relacionados con la automatización de pruebas sobre *XAML*.

Las posibilidades que ofrece *X-Aute* aplicadas a la automatización de pruebas *GUI* en interfaces *WPF* significan una gran aportación ya que es una tecnología nueva sobre la que aún quedan muchas cosas por investigar y más aún en el campo de la calidad y en sus test automatizados. Las herramientas existentes no están dedicadas únicamente a este tipo de interfaces y por tanto no podrán especializarse tanto como se hará en la herramienta *X-Aute*.

2.2 Capacidades generales

A grandes rasgos, las funcionalidades que se pretende tenga la herramienta *X-Aute* son:

- Creación y ejecución de scripts de test GUI automatizados de manera sencilla.
- Registro de cada acción que ejecuta de la batería y su resultado en un fichero de trazas.

Desde un punto de vista más detallado y funcional las capacidades que presenta *X-Aute* son:

- Creación de scripts que almacenen acciones automatizadas sobre interfaces gráficas XAML.
- Ejecución de scripts que realicen acciones automatizadas sobre las *GUI* desarrollas mediante *XAML*.
- Consulta de las acciones automatizadas que se han ejecutado y el resultado final de cada una que son registradas en un fichero de log.

- Customización de la herramienta mediante parámetros de aplicación almacenados en un fichero de configuración.

La funcionalidad ofrecida por la herramienta se puede representar de manera gráfica mediante un diagrama de casos de uso como el que se puede observar en la siguiente figura:

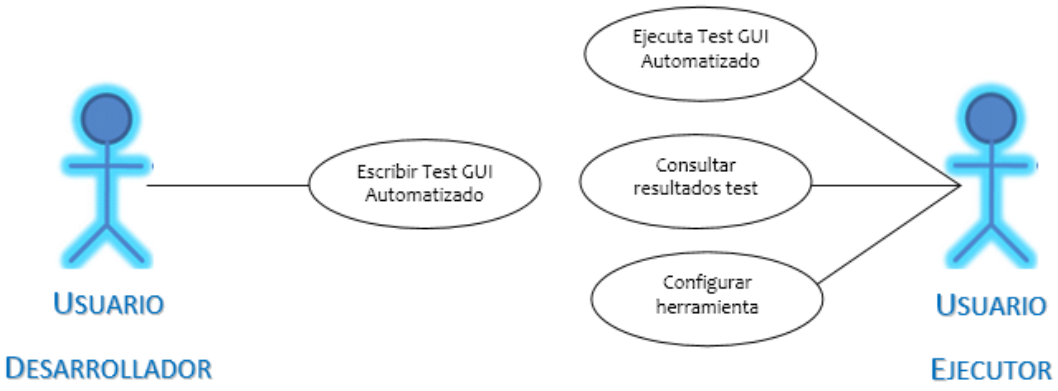


***Figura 1.** Diagrama de Casos de Uso*

2.3 Restricciones generales

Esta sección recoge todas aquellas restricciones no específicas que de alguna manera limitan el desarrollo de la herramienta.

2.3.1 Restricciones de interfaces externos

La interacción de los usuarios con la herramienta implementada se realizará a través de la plataforma de desarrollo Visual Studio 2005. Por tanto las restricciones de interfaz serán las propias del entorno de desarrollo.

2.3.2 Restricciones de rendimiento

Puesto que el objetivo final de las pruebas automatizadas es el de lograr ahorrar tiempo y recursos, incrementando además el número de zonas testeadas de las aplicaciones un objetivo será desarrollar la herramienta de modo que ofrezca el mayor rendimiento posible aunque éste siempre estará sometido al del propio entorno de desarrollo Visual Studio 2005 y al del .NET Framework 3.0 utilizado para su implementación.

2.3.3 Restricciones de desarrollo

El desarrollo de *X-Aute* se lleva a cabo mediante el seguimiento de los estándares de Ingeniería de Software propuestos por la ESA y cuya referencia es PSS-05. Los estándares presentados en la serie PSS-05 establecen un proceso de desarrollo de alta calidad, lo que permite conseguir muy buenos resultados en cuanto a fiabilidad y facilidad de mantenimiento y seguimiento del ciclo de vida del producto.

Para la implementación de la herramienta, que tiene lugar en la fase DD del ciclo de vida del proyecto, se utilizará como entorno de desarrollo el propio *Visual Studio 2005* junto al *.Net Framework 3.0* cuya tecnología permite la construcción de una herramienta de testeo automatizado de las interfaces gráficas desarrolladas con el *Framework 3.0 de .Net*, concretamente mediante la tecnología *WPF*. Así mismo también será necesaria la utilización de la herramienta *UISpy* proporcionada por el *Windows SDK* para poder observar los atributos y sus valores de los objetos de interfaz que pretendemos automatizar.

2.3.4 Restricciones tecnológicas

Para poder ejecutar la aplicación será necesario que el usuario lo haga desde un ordenador que posea las siguientes características técnicas:

- ***Sistema Operativo***: La herramienta ha de correr sobre un sistema operativo compatible con *WPF*, puesto que éste tipo de interfaz será el objetivo de su testeo, como lo son Microsoft Windows Vista© y Microsoft Windows XP©.
- ***Tarjeta gráfica***: Ya que para poder testar una *GUI XAML* es necesario que esta se esté ejecutando será imprescindible una tarjeta gráfica capaz de mostrar y hacerlo con fluidez interfaces *WPF*. Para esto será necesario una tarjeta gráfica con un mínimo de 128 MB de memoria.
- ***Monitor***: Por el mismo motivo que el comentado inmediatamente arriba el monitor ha de presentar una buena calidad de imagen, y como mínimo ha de ofrecer una resolución de 800x600 píxeles.
- ***Software Adicional***: Será necesario que el usuario haya instalado, previamente a la ejecución, el *Framework 3.0 de .Net*, ya que este es el que contiene la tecnología de Windows Presentation Foundation© (*WPF*). Además se debe disponer del entorno de desarrollo *Visual Studio 2005* instalado en la máquina. Y si el usuario quiere escribir scripts de test deberá disponer también del *Visual Studio 2005 SDK*.

Las especificaciones no mencionadas arriba se consideran secundarias y se contemplarán como aceptables todas aquellas capaces de arrancar y ejecutar sin mayores dificultades aplicaciones sobre Microsoft Windows Vista© o Microsoft Windows XP©.

2.3.5 Restricciones de usabilidad

En esta sección se estudian los requisitos concernientes tanto a los usuarios como a la usabilidad de la aplicación desarrollada.

Características cognitivas de los usuarios

Debido al ámbito de la herramienta *X-Aute* las características cognitivas de los usuarios se pueden agrupar en una sola categoría:

- ***Conocimientos Informáticos***: Todos los usuarios han de disponer los conocimientos técnicos necesarios para manejar el entorno de desarrollo *Visual Studio 2005* y estar mínimamente familiarizados con el mismo.
  - ***Usuario desarrollador***: A pesar que el desarrollo de la herramienta intentará hacer que la creación de los scripts sea lo más simple e intuitiva posible y el manual de usuario, desarrollado en la fase DD del ciclo de vida del proyecto, indique de manera clara como utilizar la herramienta y el modo en que se deben escribir las acciones, será necesario que el usuario tenga unas nociones mínimas del lenguaje de programación C# o en su defecto Java.
  - ***Usuario ejecutor:*** Este tipo de usuario han de estar familiarizados con la apertura, visualización y comprensión de documentos XML y HTML, por lo que deben saber utilizar algún editor o navegador web que les permita abrir los documentos. Además para poder realizar el mantenimiento de los script han de tener algunos conocimientos sobre C# o Java, más básicos si cabe que los de los usuario desarrolladores puesto que este trabajo se limitará a pequeñas modificaciones sobre el script creado.

Análisis de los usuarios

Se puede encontrar un análisis en profundidad de las características de los usuarios de la plataforma en la sección 2.4 Características de los Usuarios.

Análisis del entorno de utilización

El lugar en qué se utilice el software será una localización desde la que se tenga accesible para su ejecución la interfaz gráfica de usuario a testear, lo que puede abarcar desde alguna oficina o departamento de desarrollo hasta incluso lugares públicos de cualquier índole, en función de la seguridad, distribución comercial, dependencias, etc. de la aplicación WPF de la que se pretendan automatizar las pruebas

2.4 Características de los usuarios

La herramienta *X-Aute* tiene unos tipos de usuario muy claros y definidos, cada uno de los cuales tiene su propio perfil, tanto cognitivo como desde el punto de vista operacional. Estos tipos de usuario son:

- ***Usuario desarrollador***: Serán los encargados de realizar las baterías de test, formadas por múltiples casos de test que a su vez estarán formados por múltiples acciones a ejecutar de manera automatizada sobre los controles de la *GUI* y los cuales se pretenden testar mediante la posterior ejecución de dicho script a través de la herramienta.
- ***Usuario ejecutor***: Serán los encargados de ejecutar los scripts de prueba que los 'Desarrolladores' han creado y determinar si la aplicación ha pasado las pruebas o por el contrario se ha producido algún error. Para ello observarán los ficheros de log de cada batería asegurándose del correcto funcionamiento de cada acción. Además se han de encargar del mantenimiento de las baterías, es decir, si los errores producidos en la ejecución no vienen a consecuencia de algún fallo en la interfaz testeada sino que tienen como fuente del error el propio script ya sea porque se ha escrito mal o porque la interfaz ha cambiado y hay que modificar el script, ellos serán los encargados de hacer que la prueba finalice con éxito.

2.5 Entorno Operativo

El entorno operativo en el que se ejecutará la herramienta será cualquier ordenador personal, portátil o de sobremesa, que incorpore las características técnicas mínimas indicadas en el apartado 2.3.4. Restricciones Tecnológicas.

2.6 Asunciones y dependencias

Habida cuenta de lo expuesto en las secciones anteriores y de las características propias de la herramienta desarrollada *X-Aute* para este Proyecto Fin de Carrera, no es necesario determinar ningún tipo de dependencia puesto que no tendrá que interactuar con otros sistemas ya existentes.

3. Requisitos Específicos

Esta sección contiene los detalles más relevantes relacionados con la herramienta desarrollada en lo que a requisitos de usuario se refiere y que constituyen la base sobre la que se establecen las tareas de las siguientes fases del proyecto. Estos requisitos se han recogido teniendo en cuenta los casos de uso representados en el diagrama de casos de uso de la aplicación (Figura 1, página 13).

Para facilitar el tratamiento de los requisitos de usuario a lo largo de la vida del proyecto, se ha seguido el siguiente criterio de denominación:

RU [*Tipo de requisito*] – [*Número de requisito en la sección*]

Donde:

- *Tipo de requisito:* Es el distintivo de los requisitos, puede ser CAP para los requisitos de capacidad y RES para los requisitos de restricción.
- *Número de requisito en la sección:* Este número establece un orden entre los requisitos que se encuentren en la misma sección.

Además del identificador, cada uno de los requisitos de usuario está caracterizado por un conjunto de pares atributo-valor cuya semántica se describe a continuación:

- *Descripción:* Este atributo contiene un enunciado en forma de explicación para facilitar la interpretación del requisito correspondiente.
- *Necesidad:* Mide el nivel de importancia del requisito de acuerdo a los siguientes valores:
  - *Esencial:* Es imprescindible que sea implementado.
  - *Deseable:* Debería ser implementado si no conlleva una dificultad excesiva.
  - *Opcional:* Sería interesante su implementación.

- *Prioridad:* Indica en qué momento de la vida del proyecto se implementará el requisito, teniendo en cuenta el ciclo de vida iterativo evolutivo. Los posibles valores para el requisito son:
  - *N-ésima iteración:* Donde N-ésima se ha de sustituir por el ordinal correspondiente a la iteración del proyecto durante la cual se implementará el requisito de usuario.
  - *Por determinar:* Este valor denota que la iteración concreta en la que será implementado el requisito se determinará más adelante en el ciclo de vida.

- *Estabilidad:* Sirve para representar el grado en el que se espera que el requisito no sufra modificaciones durante el desarrollo del proyecto. Tiene cuatro posibles valores, que ordenados de mayor a menor son: *Muy Alta, Alta, Baja* y *Muy Baja*.

- *Claridad:* Mide cuán preciso y entendible es el requisito calificado. Su propósito es permitir el estudio de la calidad de los requisitos del proyecto Sus posibles valores, ordenados de mayor menor son: *Muy Alta, Alta, Baja* y *Muy Baja*.

- *Verificabilidad:* Al igual que ocurre con el atributo anterior, el objetivo de este atributo es la obtención de métricas de la calidad de los requisitos de usuario, pero, en este caso, midiendo la facilidad para verificar que el requisito ha sido incorporado al sistema. Los posibles valores del atributo, ordenados de manera decreciente son: *Muy Alta, Alta, Baja* y *Muy Baja*.

El resto de la sección contiene los requisitos de usuario identificados por medio del análisis de las características y de las necesidades que tiene que cubrir la plataforma.

3.1 Requisitos de Capacidad

Esta subsección enumera, de forma organizada, los requisitos de capacidad de la herramienta *X-Aute* y que describen los procesos y funcionalidades de que ha de proporcionar.

Identificador	**RU CAP-01**			
Descripción	La herramienta permitirá crear acciones automatizas sobre controles de GUI.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU CAP-01. *Crear acciones automatizadas.*

Identificador	**RU CAP-02**			
Descripción	La herramienta permitirá ejecutar las acciones automatizas sobre controles de GUI.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU CAP-02. *Ejecutar acciones automatizadas.*

Identificador	**RU CAP-03**			
Descripción	La herramienta permitirá crear scripts de pruebas que serán la unión de varias acciones automatizadas sobre los controles *GUI* de alguna funcionalidad individualizada de la interfaz. A cada uno de dichos script se les llamará *Caso de Test*.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU CAP-03.** Crear Casos de Test.*

Identificador	**RU CAP-04**			
Descripción	La herramienta permitirá ejecutar los *Casos de Test* que hayan sido previamente creados.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU CAP-04.** Ejecutar Casos de Test.*

Identificador	**RU CAP-05**			
Descripción	La herramienta permitirá crear conjuntos de pruebas que serán la unión de varios casos de test que prueben la funcionalidad completa de la interfaz o alguna parte muy grande de esta. A cada uno de dichos conjuntos se les llamará *Batería de Test*.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU CAP-05.** Crear Baterías de Test.*

Identificador	RU CAP-06			
Descripción	La herramienta permitirá ejecutar las *Baterías de Test* que hayan sido previamente creados.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU CAP-06. *Ejecutar Batería de Test.*

Identificador	RU CAP-07			
Descripción	La herramienta permitirá crear las acciones *automatizadas* sobre los controles *GUI*, los *Casos de Test* y las *Baterías de Test* de una manera sencilla e intuitiva. Sin necesidad de grandes conocimiento de programación por parte del usuario.			
Necesidad	☐ *Esencial*	☑ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Claridad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja

RU CAP-07. *Creación de scripts sencilla.*

Identificador	RU CAP-08			
Descripción	La herramienta debe hacer de manera transparente al usuario la creación e instanciación de objetos de cada uno de los controles *GUI* objetivo de las acciones automatizadas.			
Necesidad	☐ *Esencial*	☑ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU CAP-08. *Creación e instanciación de objetos transparente.*

Identificador	**RU CAP-09**			
Descripción	La herramienta debe hacer de manera transparente al usuario las búsquedas y ejecuciones de las acciones derivadas por la acción concreta que el usuario pretende se ejecute automáticamente.			
Necesidad	☐ *Esencial*	☑ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Claridad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja

***RU CAP-09.** Búsquedas de controles transparentes.*

Identificador	**RU CAP-10**			
Descripción	La herramienta debe ofrecer una plantilla que facilite la creación de los *Casos de Test*.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU CAP-10.** Plantilla para creación de Casos de Test.*

Identificador	**RU CAP-11**			
Descripción	La herramienta debe ofrecer una plantilla que facilite la creación de las *Baterías de Test*.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU CAP-11.** Plantilla para creación de Baterías de Test.*

Identificador	RU CAP-12			
Descripción	La herramienta escribirá en un fichero de log cada una de las acciones que el desarrollador ha escrito en los *Casos de Test* y son ejecutadas.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU CAP-12. *Registro de acciones en fichero log.*

Identificador	RU CAP-13			
Descripción	La herramienta escribirá en un fichero log cada una de las acciones que son provocadas por una acción que el usuario pretende automatizar pero no ha escrito directamente en el script. Estas acciones irán anidadas sobre la acción que las provoca.			
Necesidad	☐ *Esencial*	☑ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Claridad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja

RU CAP-13. *Registro de sub-acciones en fichero log.*

Identificador	RU CAP-14			
Descripción	La herramienta escribirá en el fichero de log junto a cada acción ejecutada su resultado y una descripción en caso que se haya producido un error.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU CAP-14. *Registro del resultado de la acción fichero log.*

Identificador	**RU CAP-15**			
Descripción	La herramienta escribirá en el fichero de log, si se desea, junto a cada acción ejecutada el tiempo que ha llevado su ejecución total (incluyendo el tiempo de las acciones que haya podido provocar). Existirá un parámetro de aplicación que permitirá indicar a la herramienta si se desea calcular el tiempo que comprende cada acción o no.			
Necesidad	☐ *Esencial*	☐ Deseable	☑ Opcional	
Prioridad	Por determinar			
Estabilidad	☐ *Muy Alta*	☐ Alta	☑ Baja	☐ Muy Baja
Claridad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU CAP-15.** Parámetro de registro del tiempo de la acción en el fichero log.*

Identificador	**RU CAP-16**			
Descripción	La herramienta escribirá el fichero de log con formato *XML*.			
Necesidad	☐ *Esencial*	☑ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU CAP-16.** Fichero de log con formato XML.*

Identificador	**RU CAP-17**			
Descripción	La herramienta obtendrá un nuevo fichero de log en formato *HTML*, utilizando para ello una hoja *XSLT* sobre el *XML*.			
Necesidad	☐ *Esencial*	☑ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU CAP-17.** Hoja de transformación XSLT.*

Identificador	RU CAP-18			
Descripción	En el fichero *HTML*, se mostrarán primero los resultados globales de los *Casos de Test*, ordenados en una tabla, *y de la Batería*. El nombre de cada *Caso de Test* será un enlace que dirige a los resultados pormenorizados del caso acción por acción.			
Necesidad	☐ *Esencial*	☑ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Claridad	☐ *Muy Alta*	☑ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU CAP-18. *Mostrado de datos en fichero HTML.*

Identificador	RU CAP-19			
Descripción	La aplicación tendrá los siguientes parámetros configurables: Si se desea contabilizar el tiempo de cada acción y almacenarlo en el fichero log, la ruta donde dejar dicho fichero y la ruta donde encontrar la hoja de transformación.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU CAP-19. *Parámetros configurables.*

Identificador	RU CAP-20			
Descripción	Los parámetros configurables quedarán almacenados en un fichero de configuración y serán cargados al inicio de la ejecución de cada *Batería*.			
Necesidad	☐ *Esencial*	☐ Deseable	☑ Opcional	
Prioridad	Por determinar			
Estabilidad	☐ *Muy Alta*	☐ Alta	☑ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU CAP-20. *Fichero de configuración.*

3.2 Requisitos de Restricción

En esta sección se enumeran y especifican los requisitos que delimitan y restringen el proceso y las características del desarrollo de *X-Aute*.

3.2.1 Requisitos de Interfaces Externas

Identificador	**RU RES-01**			
Descripción	El interfaz de usuario de la aplicación a través de la cual se accede a la herramienta está basada en las características del interfaz propia del entorno de desarrollo Microsoft ® Visual Studio 2005.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU RES-01.** Restricción de interfaz.*

3.2.2 Requisitos de Hardware

Identificador	**RU RES-02**			
Descripción	La herramienta deberá ejecutarse bajo el IDE de Visual Studio 2005 en ordenadores personales.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU RES-02.** Restricción de hardware.*

3.2.3 Requisitos de Software

Identificador	RU RES-03			
Descripción	La herramienta se ejecutará en ordenadores con Microsoft *Windows Vista®* o *Microsoft Windows XP®*.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU RES-03. *Restricción de software (I).*

Identificador	RU RES-04			
Descripción	La herramienta se ejecutará en ordenadores que disponen del entorno de desarrollo *Microsoft® Visual Studio 2005*.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU RES-04. *Restricción de software (II).*

Identificador	RU RES-05			
Descripción	La herramienta se ejecutará en ordenadores que disponen del *Microsoft® .Net Framework 3.0*.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

RU RES-05. *Restricción de software (III).*

Identificador	**RU RES-06**			
Descripción	La herramienta se ejecutará en ordenadores que disponen del kit de desarrollo software *Microsoft Windows SDK®*.			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	1ª Iteración			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU RES-06.** Restricción de software (IV).*

3.2.4 Requisitos de Desarrollo

Identificador	**RU RES-07**			
Descripción	La herramienta deberá desarrollarse tomando como base los estándares de ingeniería de software propuestos por la ESA (ESA Board for Software Standardisation and Control, 1996) y (ESA Board for Software Standardisation and Control, 1995).			
Necesidad	☑ *Esencial*	☐ Deseable	☐ Opcional	
Prioridad	Por determinar			
Estabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Claridad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja
Verificabilidad	☑ *Muy Alta*	☐ Alta	☐ Baja	☐ Muy Baja

***RU RES-07.** Restricción de desarrollo.*

4. Lista de Requisitos de Usuario

SECCIÓN
VI
BIBLIOGRAFÍA

15. BIBLIOGRAFÍA

Adeoti-Adekeye, W. (1997). *The importance of Management information systems.* MCB UP Ltd .

Applegate, L., McFarlan, F., & McKenney, J. (1996). *Corporate Information Systems Management: Text and Cases.* McGraw-Hill.

Barreau, D. (2001). *The hidden costs of implementing and maintaining information systems .* MCB UP Ltd.

Borland Software - http://www.borland.com. (s.f.). *Borland SilkTest*. Obtenido de Borland Products: http://www.borland.com/us/products/silk/silktest/index.html

Buades, G. (2002). *Calidad en Ingeniería del Software*. Obtenido de http://dmi.uib.es/~bbuades/calidad/

Chappell, D. (23 de Octubre de 2006). *MSDN España*. (Chappell & Associates) Obtenido de Introducción a Windows Presentation Foundation: http://www.microsoft.com/spanish/msdn/articulos/archivo/231006/voices/introducingwpf.mspx

De Pablos Heredero, C., Izquierdo Loyola, V., López-Hermoso, J., Martín-Romo Romero, S., Montero Navarro, A., & Nájera Sánchez, J. (2001). *Dirección y Gestión de los Sistemas de Información en la Empresa.* Madrid: ESIC.

Duff, N., & Assad, M. (1980). *Information Management: An Executive Approach.* Oxford: Oxford University Press.

Dustin, E., Rashka, J., & Paul, J. (1999). *Automated Software Testing "Introduction, management and performance".* Addison-Wesley.

ESA Board for Software Standardisation and Control. (1991). *ESA Software Engineering Standars* (Vol. 2). European Space Agency.

ESA Board for Software Standardisation and Control. (1996). *Guide to applying the ESA software engineering standards to small software projects.* European Space Agency.

Fewster, M. &. (1999). *Software Test Automation.* Gran Bretaña: Addison-Wesley.

IBM - Rational Robot - http://www-01.ibm.com/software/awdtools/tester/robot. (s.f.). Obtenido de IBM: http://www-01.ibm.com/software/awdtools/tester/robot/features/index.html

Kaner, C., Falk, J., & Nguyen, H. Q. (1993). *Testing Computer Software* (Segunda ed.). ITP.

Katrib, M., Del Valle, M., Sierra, I., & Hernández, Y. (2007). *Windows Presentation Foundation.* Madrid: Netalia.

Langemo, M. (1980). *Records management/word procession-a needed team effort.*

Laudon, K., & J.P., L. (2002). *Sistemas de Información gerencial: Organización y Tecnología de la empresa conectada a la red.* México: Pearson Educación.

Laudon, K., & Laudon, J. (1998). *Management Information Systems: New Approaches to Organization and Technology.* Londres: Prentice Hall International.

Microsoft Corp. (6 de Noviembre de 2006). *Wikipedia*. Obtenido de .Net: http://es.wikipedia.org/wiki/.Net

Microsoft Corporation. (2005). Obtenido de Programa Microsoft para Universidades: http://www.ms-universidades.com/sites/universidades/msdn-academic.aspx

Microsoft Corporation. (2007). *MDSN*. Retrieved from XAML Overview: http://msdn2.microsoft.com/en-us/library/ms752059.aspx

Microsoft Corporation. (2006). *Microsoft .Net Framework*. Retrieved from Windows CardSpace: http://cardspace.netfx3.com/

Microsoft Corporation. (2006). *Microsoft .Net Framework*. Retrieved from Windows Comunication Foundation: http://wcf.netfx3.com/

Microsoft Corporation. (2006). *Microsoft .Net Framework*. Retrieved from Windows Workflow Foundation: http://wf.netfx3.com/

Microsoft Corporation. (2007). *MSDN*. Retrieved from What Is Windows Communication Foundation: http://msdn2.microsoft.com/en-us/library/ms731082.aspx

Microsoft Corporation. (26 de Octubre de 2006). *Wikipedia*. Obtenido de XAML: http://es.wikipedia.org/wiki/XAML

Moroney, L. (19 de Febrero de 2007). *MSDN España*. Obtenido de Introducción a Silverlight: http://www.microsoft.com/spanish/msdn/articulos/archivo/010507/voices/bb404300.mspx

Panello, D. (s.f.). *Introducción a Microsoft .Net*. Obtenido de http://www.dcp.com.ar/intro_net.htm

Wikipedia - HP QuickTest Professional . (s.f.). Obtenido de Wikipedia: http://en.wikipedia.org/wiki/HP_QuickTest_Professional